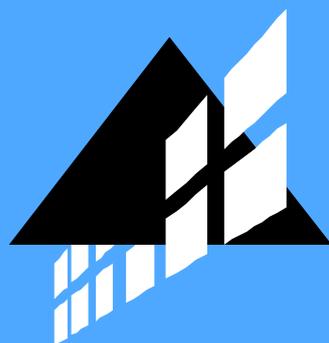
Manual de Programação ST
MasterTool Extended Edition
MT8000

Rev. C 08/2010

Cód. Doc.: MP399003

Conforme o Código de Defesa do Consumidor vigente no Brasil, informamos, a seguir, aos clientes que utilizam nossos produtos aspectos relacionados com a segurança de pessoas e instalações.

Os equipamentos de automação industrial fabricados pela Altus são robustos e confiáveis devido ao rígido controle de qualidade a que são submetidos. No entanto, equipamentos eletrônicos de controle industrial (controladores programáveis, comandos numéricos, etc.) podem causar danos às máquinas ou processos por eles controlados em caso de defeito em suas partes e peças ou de erros de programação ou instalação, podendo inclusive colocar em risco vidas humanas.

O usuário deve analisar as possíveis consequências destes defeitos e providenciar instalações adicionais externas de segurança que, em caso de necessidade, sirvam para preservar a segurança do sistema, principalmente nos casos da instalação inicial e de testes.

Os equipamentos fabricados pela Altus não trazem riscos ambientais diretos, não emitindo nenhum tipo de poluente durante sua utilização. No entanto, no que se refere ao descarte dos equipamentos, é importante salientar que quaisquer componentes eletrônicos incorporados em produtos contêm materiais nocivos à natureza quando descartados de forma inadequada. Recomenda-se, portanto, que quando da inutilização deste tipo de produto, o mesmo seja encaminhado para usinas de reciclagem que deem o devido tratamento para os resíduos.

É imprescindível a leitura completa dos manuais e/ou características técnicas do produto antes da instalação ou utilização do mesmo.

Os exemplos e figuras deste documento são apresentados apenas para fins ilustrativos. Devido às possíveis atualizações e melhorias que os produtos possam incorrer, a Altus não assume a responsabilidade pelo uso destes exemplos e figuras em aplicações reais. Os mesmos devem ser utilizados apenas para auxiliar na familiarização e treinamento do usuário com os produtos e suas características.

A Altus garante os seus equipamentos conforme descrito nas Condições Gerais de Fornecimento, anexada às propostas comerciais.

A Altus garante que seus equipamentos funcionam de acordo com as descrições contidas explicitamente em seus manuais e/ou características técnicas, não garantindo a satisfação de algum tipo particular de aplicação dos equipamentos.

A Altus desconsiderará qualquer outra garantia, direta ou implícita, principalmente quando se tratar de fornecimento de terceiros.

Os pedidos de informações adicionais sobre o fornecimento e/ou características dos equipamentos e serviços Altus devem ser feitos por escrito. A Altus não se responsabiliza por informações fornecidas sobre seus equipamentos sem registro formal.

DIREITOS AUTORAIS

Sumário

1. Introdução

A linguagem ST (Structured Text) é uma linguagem textual estruturada, de alto nível, com recursos semelhantes às linguagens C e Pascal. Com ela pode-se escrever programas com os comandos IF, THEN, ELSE, laços FOR e WHILE, criar variáveis locais e vetores, criar e chamar sub-rotinas, etc. Com possibilidade de acesso aos operandos do CP, é tida como uma alternativa ao uso da linguagem gráfica Ladder.

Esta facilidade, programação em linguagem ST, é fornecida com MasterTool XE Advanced, sendo disponibilizada para uso com a UCP AL-2004, bem como para as UCPs da Série Ponto. Através desta linguagem é possível a criação de módulos procedimento (módulo P) e módulos função (módulo F), conforme definido pela norma IEC-1131-3.

Após a criação, os módulos ST possuem as mesmas características dos demais módulos F ou P já existentes. O número de parâmetros de entrada e saída pode ser configurado. A chamada do novo módulo ST deve ser feita no programa aplicativo Ladder com a instrução CHF ou CHP, tal como já é feita para os demais módulos F e módulos P.

O módulo criado em linguagem ST ainda pode ser lido de volta do CP para o micro. Neste caso, o programa fonte ST é restaurado normalmente, inclusive com os comentários. Existem senhas de proteção contra leitura e/ou modificação não autorizada, protegendo a tecnologia do desenvolvedor. O MasterTool XE implementa um subconjunto da Linguagem ST, sendo que engloba a maioria dos comandos e estruturas de dados e comandos definidos pela IEC-61131-3.

O desempenho de execução no CP de um módulo ST é melhor que um módulo equivalente em ladder, pois o módulo ST é verificado como um todo, enquanto que em ladder ele é dividido em várias chamadas de instruções.

Os principais benefícios da utilização da linguagem ST são:

- Outra opção de linguagem de programação, padronizada por norma internacional
- Possibilidade da criação de algoritmos mais complexos e utilização de software estruturado
- Linguagem textual em vez de gráfica: pode-se usar copia/cola/substitui ou macros de editores de texto tradicionais
- Menor tempo de desenvolvimento, implicando em menor custo de engenharia
- Melhor desempenho de execução

A figura, a seguir, traz um exemplo de utilização da Linguagem ST no MasterTool XE:

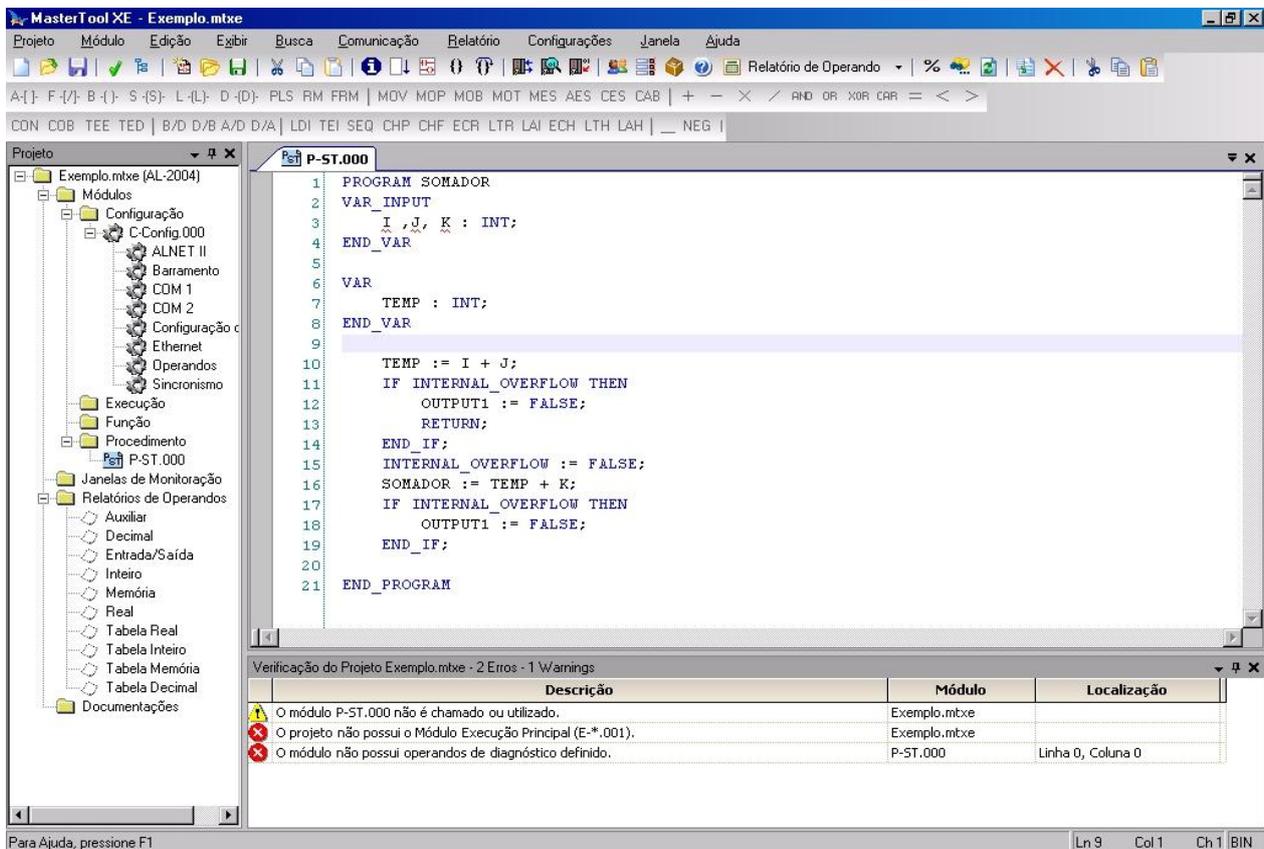



Figura 1-1. Exemplo de utilização da linguagem ST

Lite, Professional e Advanced

O software MasterTool XE, possui três versões de distribuição, cada uma com um perfil otimizado de acordo com a necessidade. São elas:

- **Lite**: software programador específico para pequenas aplicações. Esta versão não suporta ST.
- **Professional**: software programador contendo as ferramentas necessárias para todas as linhas de CPs da Altus.
- **Advanced**: software programador com ferramentas para aplicações de grande porte.

Cada uma destas versões possui características, finalidades e funcionalidades específicas para cada propósito. O detalhamento sobre as diferenças entre as versões pode ser visto no Manual de Utilização do MasterTool XE.

Documentos Relacionados a este Manual

Para obter informações adicionais sobre o uso da Linguagem ST podem ser consultados outros documentos (manuais e características técnicas) além deste. Estes documentos encontram-se disponíveis em sua última revisão em www.altus.com.br.

Cada produto possui um documento denominado Característica Técnica (CT), onde se encontram as características do produto em questão. Adicionalmente o produto pode possuir Manuais de Utilização (os códigos dos manuais são citados nas CTs).

Aconselhamos os seguintes documentos como fonte de informação adicional:

- Características Técnicas MT8000
- Manual de Utilização do MasterTool XE
- Manual de Programação do MasterTool XE

Inspeção Visual

Antes de proceder à instalação, é recomendável fazer uma inspeção visual cuidadosa dos produtos, verificando se não há danos causados pelo transporte. Deve ser verificado se todos os componentes de seu pedido estão em perfeito estado. Em caso de defeitos, informe a companhia transportadora e o representante ou distribuidor Altus mais próximo.

É importante registrar o número de série de cada produto recebido, bem como as revisões de software, caso existentes. Essas informações serão necessárias caso se necessite contatar o Suporte Técnico da Altus.

Suporte Técnico

Para entrar em contato com o Suporte Técnico da Altus em São Leopoldo, RS, ligue para +55-51-3589-9500. Para conhecer os centros de Suporte Técnico da Altus existentes em outras localidades, consulte nosso site (www.altus.com.br) ou envie um e-mail para altus@altus.com.br.

Se o equipamento já estiver instalado, tenha em mãos as seguintes informações ao solicitar assistência:

- Os modelos dos equipamentos utilizados e a configuração do sistema instalado.
- O número de série da UCP.
- A revisão do equipamento e a versão do software executivo, constantes na etiqueta afixada na lateral do produto.
- Informações sobre o modo de operação da UCP, obtidas através do programador MasterTool.
- O conteúdo do programa aplicativo (módulos), obtido através do programador MasterTool.
- A versão do programador utilizado.

Mensagens de Advertência Utilizadas neste Manual

Neste manual, as mensagens de advertência apresentarão os seguintes formatos e significados:

PERIGO:
Relatam causas potenciais, que se não observadas, *levam* a danos à integridade física e saúde, patrimônio, meio ambiente e perda da produção.

CUIDADO:
Relatam detalhes de configuração, aplicação e instalação que *devem* ser seguidos para evitar condições que possam levar a falha do sistema e suas conseqüências relacionadas.

ATENÇÃO:
Indicam detalhes importantes de configuração, aplicação ou instalação para obtenção da máxima performance operacional do sistema.

2. Descrição Técnica

Esta seção apresenta as características técnicas do produto Editor ST.

Características de Software

Tipos de Dados

Os tipos de dados disponíveis são mostrados na tabela a seguir:

	Descrição	Bits	Operando do CP Associado
BOOL	Booleano	1	Bit de %A, %E, %S ou %M
BYTE	Seqüência de 8 bits	8	Operando %A, %E ou %S
WORD	Seqüência de 16 bits	16	Operando %M ou %TM
DWORD	Seqüência de 32 bits	32	Operando %I ou %TI
USINT	Inteiro curto não sinalizado	8	Operando %A, %E ou %S
INT	Inteiro	16	Operando %M ou %TM
DINT	Inteiro longo	32	Operando %I ou %TI
REAL	Real	32	Operando %F ou %TF

Tabela 2-1. Tipos de dados

Uma variável é uma área de memória que armazena um valor definido pelo seu tipo de dado. Todas as variáveis devem ser declaradas antes de serem utilizadas. Seu escopo é limitado à função ou ao programa em que foram declarados, permitindo que os nomes possam ser reutilizados em outras partes do software, sem que ocorra nenhum conflito.

As variáveis declaradas podem estar associadas a operandos do controlador programável, possibilitando, assim, a criação de algoritmos para execução do controle do processo industrial desejado.

Limites de Software

	Descrição
Chamadas aninhadas de funções	16 chamadas de função
Tamanho do módulo P ou F	Até 32KB
Área de dados para variáveis	Até 3KB[1]
Quantidade de funções	O número de funções é limitado pelo tamanho do módulo.

Tabela 2-2. Limites de software

Operandos do CP

Um módulo desenvolvido utilizando a Linguagem ST pode acessar os operandos do CP. A tabela, a seguir, apresenta os operandos do CP, bem como sua compatibilidade com a linguagem ST:

[1] Para variáveis declaradas entre VAR..END_VAR que não estão associadas a operandos do CP, e para variaveis internas alocadas automaticamente pelo MasterTool XE e utilizadas em operações temporárias.

Tipo de Operando	Pode ser acessado no ST ?
%A, %E e %S	Sim
%M e %TM	Sim
%I e %TI	Sim
%F e %TF	Sim
%D e %TD	Não

Tabela 2-3. Tipos de operandos

As variáveis utilizadas nos módulos ST devem estar declaradas antes de serem utilizadas. Não se pode utilizar operandos do CP diretamente na função ST, mas é possível mapear estes endereços nas variáveis declaradas da função.

Operadores

Os operadores de expressões aritméticas e lógicas em ST são semelhantes aos de outras linguagens, conforme mostra na tabela a seguir:

Tipo	Operadores	
Matemáticos	+	Soma
	-	Subtração
	*	Multiplicação
	/	Divisão
	MOD	Resto
Lógicos	& AND	Operação “E”
	OR	Operação “OU”
	XOR	Operação “OU” exclusive
	NOT	Operação “NÃO”
Comparação	<	Menor que
	>	Maior que
	<=	Menor ou igual
	>=	Maior ou igual
Igualdade	=	Igualdade
	<>	Diferença

Tabela 2-4. Operandos e expressões aritméticas

Comandos

A tabela, a seguir, apresenta os comandos da linguagem ST:

Tipo	Comandos
Repetição	WHILE REPEAT FOR
Seleção	IF CASE

Tabela 2-5. Comandos da linguagem ST

Itens não Implementados da Norma IEC 61131-3

Os itens listados, a seguir, fazem parte das definições existentes na norma IEC 61131-3 e não estão implementados neste produto:

- Tipos de dado SINT, UINT, UDINT, LREAL
- Tipos de dado TIME, DATE e STRING
- *Enumerated*, *Sub-Range*
- *Arrays* de mais de uma dimensão (matrizes)
- *Structs*
- *Function Blocks*
- *Resource*, *Tasks*, *Configurations*

3. Procedimentos

Nesta seção é apresentado exemplos de como gerar um módulo procedimento ou um módulo função com o Editor ST. Em ambos, serão utilizados um programa de filtro digital que deve ser aplicado sobre três operandos de entrada.

Criando um Módulo P ou F em Linguagem ST

Depois de aberto o Master Tool XE e criado um novo projeto deve ser selecionada a opção no menu **Módulo /Novo**. A seguinte tela será mostrada.

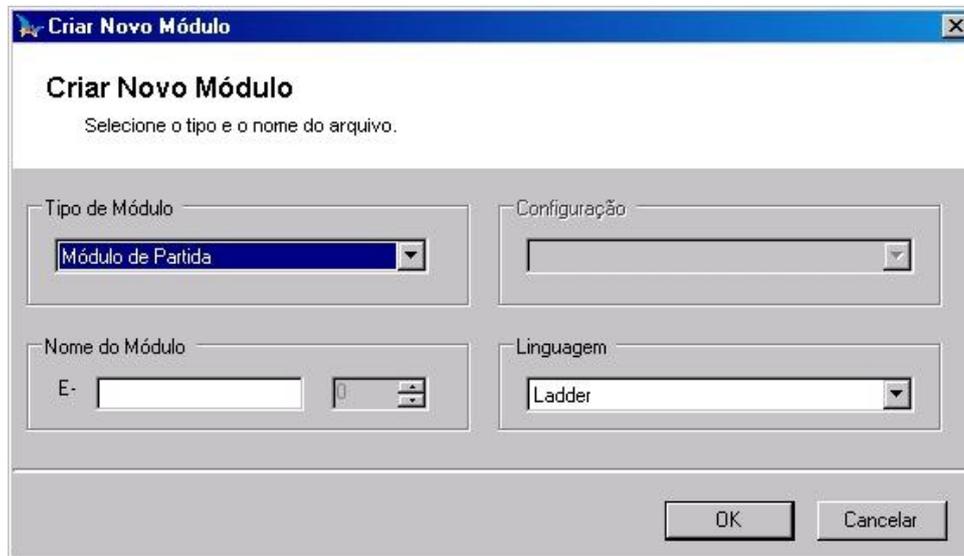



Figura 3-1. Criando um módulo ST

Selecione uma das seguintes opções em "Tipo de Módulo": "**Módulo Função**" ou "**Módulo Procedimento**". Na caixa de seleção "Linguagem" selecione "**ST**". O Editor ST será executado e estará pronto para a edição do módulo escolhido após a seleção do botão OK.

A seguir, um exemplo de uma possível configuração para um novo módulo de Função na Linguagem ST:

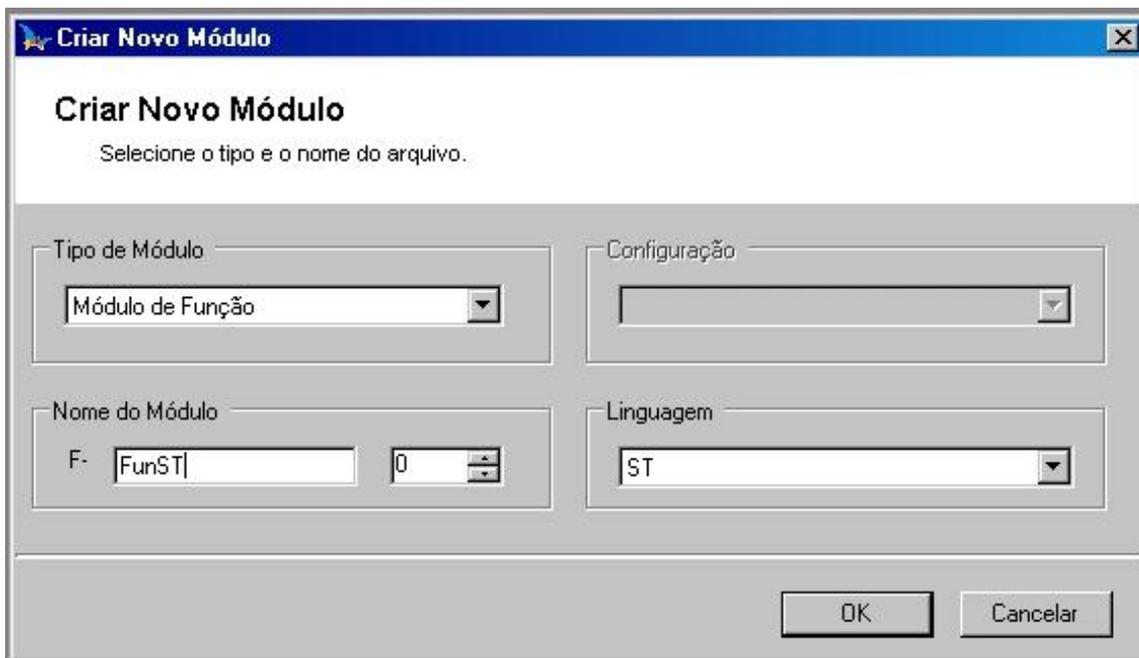



Figura 3-2. Criando um novo módulo ST

Declarando Variáveis Globais

Maiores detalhes sobre a declaração e os tipos de variáveis podem ser encontrados na seção **Programação,** no capítulo Variáveis. Neste exemplo, são declaradas as constantes do filtro como constantes de uso global.

```
VAR CONSTANT
TEMPO_FILTRO:          REAL:=0.1;     (*Constante de tempo do filtro*)
TEMPO_AMOSTRAGEM:      REAL:=0.02;    (*Período de amostragem*)
END_VAR
```

Criando uma Função para Implementar um Filtro

Neste exemplo, são filtrados mais de um valor de entrada e para tanto se torna necessário o uso de uma função. O papel dessa função é aplicar um filtro cuja constante de tempo e o tempo de amostragem estão definidos como variáveis globais. Trata-se de um filtro recursivo e, portanto, além do valor de entrada (ENTRADA), é necessário um parâmetro com o último valor filtrado (SAIDA_ANTERIOR).

```
VAR CONSTANT
TEMPO_FILTRO:          REAL:=0.1;     (*Constante de tempo do filtro*)
TEMPO_AMOSTRAGEM:      REAL:=0.02;    (*Período de amostragem*)
END_VAR

FUNCTION FILTRAR:       REAL

VAR_INPUT
        ENTRADA:                INT;          (*Dado de entrada a ser filtrado*)
        SAIDA_ANTERIOR:         REAL;         (*Resultado anterior da filtragem*)
END_VAR

VAR
        FATOR_FILTRO:           REAL;         (*Variável local para o fator do filtro*)
END_VAR

        (*Guarda a razão do tempo de amostragem pela constante de tempo do*)
        (*filtro em uma variável local*)
        FATOR_FILTRO := TEMPO_AMOSTRAGEM/TEMPO_FILTRO;

        (*Aplica o filtro sobre a entrada*)
        FILTRAR := SAIDA_ANTERIOR + (ENTRADA - SAIDA_ANTERIOR)*FATOR_FILTRO;

END_FUNCTION
```

Escrevendo o Código Principal – PROGRAM – para um Módulo P

O objetivo deste código é aplicar o filtro definido pelo função FILTRAR sobre três operandos de entrada (%M200, %M201 e %M202). Para tanto é necessário guardar os resultados obtidos na última filtragem e, como o resultado de cada chamada do filtro será o resultado anterior na chamada seguinte, podem ser os mesmos operandos (%F0, %F1 e %F2).

```
PROGRAM FILTRO
VAR
       ENTRADA_0              AT %M200:    INT;
       ENTRADA_1              AT  %M201:    INT;
       ENTRADA_2              AT  %M202:    INT;
       ULTIMO_RESULTADO_0     AT  %F0:      REAL;
       ULTIMO_RESULTADO_1     AT  %F1:      REAL;
       ULTIMO_RESULTADO_2     AT  %F2:      REAL;
       RESULTADO_0            AT  %F0:      REAL;
       RESULTADO_1            AT  %F1:      REAL;
       RESULTADO_2            AT  %F2:      REAL;
END_VAR

(*Aplica filtro sobre as entradas*)
RESULTADO_0:=FILTRAR(ENTRADA_0, ULTIMO_RESULTADO_0);
RESULTADO_1:=FILTRAR(ENTRADA_1, ULTIMO_RESULTADO_1);
RESULTADO_2:=FILTRAR(ENTRADA_2, ULTIMO_RESULTADO_2);

END_PROGRAM
```

Escrevendo o Código Principal – PROGRAM – para um Módulo F

No caso de módulo F, pode-se declarar os operandos de entrada e saída através dos parâmetros de entrada. Neste exemplo são 6 parâmetros de entrada (%M200, %M201, %M202, %F0, %F1, %F2).

Aqui surge uma característica importante da passagem de parâmetros, onde os três primeiros operandos são operandos exclusivamente de entrada, mas os três últimos são operandos de entrada e saída porque, tratando-se de um filtro recursivo, o último resultado será utilizado como entrada da iteração seguinte. Todos deverão ser declarados na área de entrada da CHF e discriminados em entrada e/ou entrada e saída na respectiva declaração no módulo ST. Para maiores detalhes vide o capítulo Passagem de Parâmetros para um Módulo F na seção Programação.

```
PROGRAM FILTRO
VAR_INPUT
       OP_ENTRADA_0:          INT;
       OP_ENTRADA_1:          INT;
       OP_ENTRADA_2:          INT;
END_VAR

VAR_IN_OUT
       OP_ENTRADA_SAIDA_0:    REAL;
       OP_ENTRADA_SAIDA_1:    REAL;
       OP_ENTRADA_SAIDA_2:    REAL;
END_VAR

(*Aplica filtro sobre as entradas*)
OP_ENTRADA_SAIDA_0:=FILTRAR(OP_ENTRADA_0, OP_ENTRADA_SAIDA_0);
OP_ENTRADA_SAIDA_1:=FILTRAR(OP_ENTRADA_0, OP_ENTRADA_SAIDA_1);
OP_ENTRADA_SAIDA_2:=FILTRAR(OP_ENTRADA_0, OP_ENTRADA_SAIDA_2);

END_PROGRAM
```

Operandos de Diagnóstico

Os operandos de diagnósticos são utilizados para reportar ao usuário os erros em tempo de execução. Os operandos podem ser configurados através do menu *Módulo / Operandos / Diagnóstico.* Caso os operandos não sejam configurados é gerado erro de compilação. Os códigos dos erros podem ser vistos na seção **Erros em Tempo de Execução**.

Para poder configurar os Operandos Temporários, o módulo ST deve estar aberto e ativo (o cursor deve estar ativo no módulo).

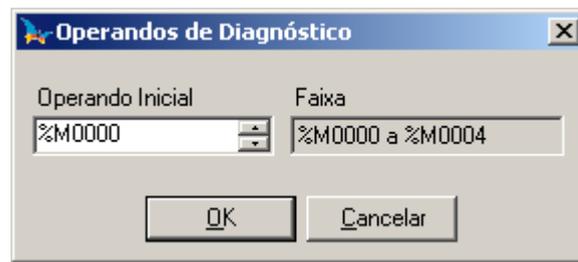



Figura 3-3. Janela de configuração dos operandos de diagnósticos

Os operandos de diagnósticos podem ser os mesmos para todos módulos ST.

Operandos Temporários

Operandos temporários são necessários para resolver algumas operações que utilizem variáveis do tipo DINT e REAL. Sua configuração não é obrigatória, mas se o compilador necessitar destes operandos será gerado erro de compilação. Os operandos podem ser configurados através do menu *Módulo / Operandos / Temporários.* A quantidade máxima de operandos, se requerida, é de 4 operandos.

Para poder configurar os Operandos Temporários, o módulo ST deve estar aberto e ativo (o cursor deve estar ativo no módulo).

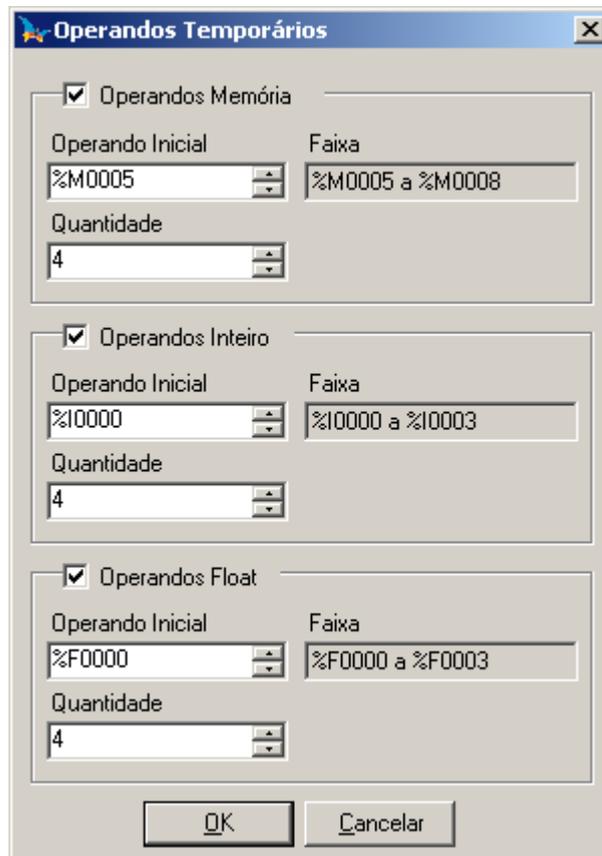



Figura 3-4. Janela de configuração dos operandos temporários

Os operandos temporários podem ser os mesmos para todos módulos ST.

Verificando o Código

Antes de enviar o módulo ao CP, o usuário deve verificar o programa, isto é, certificar-se de que não existem erros de programação no módulo. Para tanto deve utilizar o comando verificar do menu módulo. Caso existam erros, estes serão mostrados na parte inferior da janela.

Salvando o Código

Sempre que um código escrito na linguagem ST for salvo em disco, ele será verificado. O código fonte sempre é salvo no módulo, mas se a verificação acusar erros nenhum código executável é adicionado ao módulo.

> ATENÇÃO:
> O MasterTool XE só permite enviar módulos sem erros para o CP. Porém, nas versões anteriores ao MasterTool XE não executam esta consistência, neste caso um módulo com erro poderá ser enviado para o CP. Todavia, devido ao erro de verificação, o módulo não executará e será gerado um erro em tempo de execução.

Utilizando um Módulo em ST

O módulo ST é utilizado da mesma forma que outro módulo em ladder. Deve ser chamada a partir de outro módulo ladder utilizando as instruções CHP (para módulos P) ou as instruções CHF (para módulos F).

Figura 3-5. Chamada do módulo P escrito em ST

4. Programação

Este capítulo descreve as regras para escrever um programa em linguagem ST, apresentando a sua sintaxe, regras semânticas e a estrutura do código fonte.

Estrutura de um Módulo em Linguagem ST

Um módulo em linguagem ST é composto por:

- Variáveis globais (não obrigatória)
- Funções (não obrigatória)
- Programa principal

Sua estrutura básica deve assumir a seguinte forma:

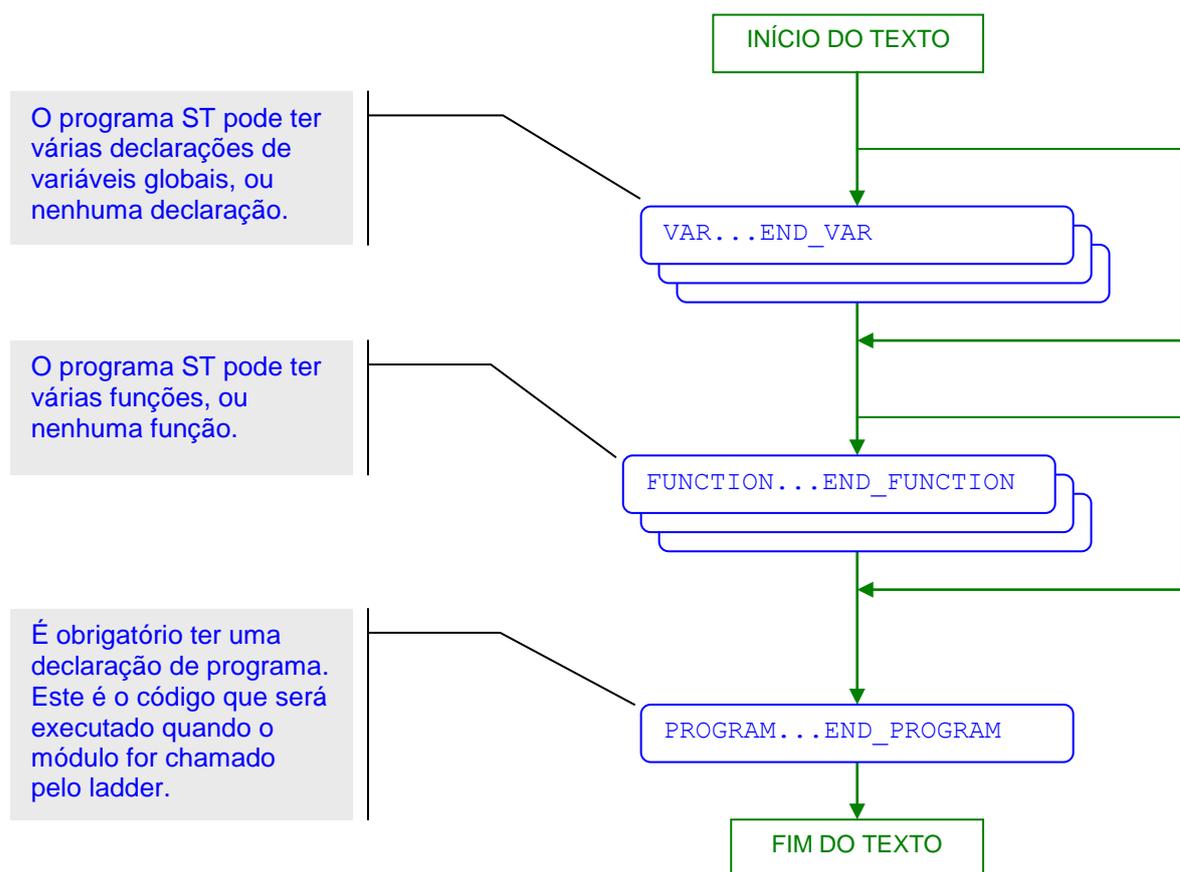



Figura 4-1. Estrutura de um programa em linguagem ST

Elementos da Linguagem ST

Os elementos da Linguagem ST são todos os símbolos, números, operadores e outras pontuações utilizadas pela linguagem e que não podem ser textualmente divididos. Os elementos da linguagem ST são:

- Identificadores
- Constantes numéricas

- Constantes booleanas
- Espaço em branco
- Comentários
- Operadores e outros sinais de pontuação

A tabela, a seguir, mostra algumas notações utilizados para definir os elementos;

Letra	qualquer letra de 'a' a 'z', maiúscula ou minúscula
Digito	qualquer digito de '0' a '9'
digito binário	qualquer digito de '0' a '1'
digito octal	qualquer digito de '0' a '7'
digito hexadecimal	qualquer digito de '0' a '9', ou qualquer letra 'a' a 'f', maiúscula ou minúscula

Tabela 4-1. Notações para representar valores e dados

Identificadores

Um identificador é usado para nomear diferentes elementos dentro da linguagem, sendo um elemento único dentro do seu escopo. É formado por letras, números e pelo caractere subscrito ("_"). Deve começar por letra ou pelo caractere subscrito, mas não pode ter dois caracteres de subscrito consecutivos e também não pode terminar com o caractere de subscrito.

Apenas os primeiros 32 caracteres da seqüência terão significado os demais serão ignorados. O uso de letras maiúsculas ou minúsculas não tem significado no identificador, ou seja, Nivel_Vaso1, nivel_vaso1 ou Nivel_vaso1 são identificadores considerados iguais.

A tabela, a seguir, traz alguns exemplos de indicadores válidos e inválidos:

Identificadores válidos	• `EstadoBomba` • `ALM_PresaoAlta` • `B1_ATIVADA` • `_1001_1` • `testePressao`
Identificadores inválidos	• `__Teste` • `Bomba__B3214` • `Valvula-125201` • `154_Ligar` • `Pressao_`

Tabela 4-2. Identificadores

ATENÇÃO:
Um identificador pode ter até 32 caracteres. Os caracteres acima deste limite serão desconsiderados, ou seja, não são significativos. O nome do identificador não é sensível a letras maiúsculas e minúsculas.

Espaço em Branco

Os caracteres de espaço, tabulação e quebra de linha são considerados espaço em branco, ou seja, podem ser utilizados livremente entre os demais elementos da Linguagem ST.

Comentários

Comentários no código fonte devem ser feitos entre "(*" e "*)".

Podem ser comentadas várias linhas com o mesmo bloco. Contudo não é permitido o uso de comentários aninhados como: "(* ... (* ... *) ... *)".

Constantes Numéricas

Existem dois tipos de constantes: inteiras e reais. Em ambos os casos, um número pode conter diversos caracteres de subscritos dentro dele. Este caractere não possui significado, servindo apenas para tornar o número mais legível. Porém o caractere de subscrito não poderá iniciar e nem terminar um número, sendo que também não é permitido o uso de dois caracteres de subscritos juntos.

```
Um número inteiro também pode ser expresso em base binária, octal ou hexadecimal. Para
isto deve ser utilizado um dos prefixos 2#, 8# e 16#  antes do número, conforme pode ser
visto nos exemplos abaixo:

2#0000_1001  (* Constante binária igual a 9   *)
2#1111_1111  (* Constante binária igual a 255 *)
8#457         (* Constante octal igual a 303 *)
16#00AA       (* Constante hexadecimal igual a 170 *)
16#8000       (* Constante hexadecimal igual a 32768 *)
```

A tabela, a seguir, traz alguns exemplos de literais numéricos válidos e inválidos:

	Exemplo	Valor em decimal
Literais numéricos válidos	• 12547 • 10_531 • -1_532 • 6978 • 2#0000_1001 • 2#01001100 • 2#001 • 8#457 • 8#6254 • 16#4576_ab4f • 16#980ABE4D • 16#FFFF_ffff	• 12547 • 10531 • -1532 • 6978 • 9 • 76 • 1 • 303 • 3244 • 1165405007 • 2550840909 • 4294967295
Literais numéricos inválidos	• _1546 • 4578_ • -_5447 • 10__135 • #0010001 • 2#4577_0158#00159 • 16#_1789 • 16#4587_	

Tabela 4-3. Literais numéricos

Literais numéricos reais contém o ponto decimal "." entre a parte inteira e fracionária. Opcionalmente, pode-se especificar o expoente da base 10 utilizando os prefixos 'E' ou 'e'. Caso isto não seja informado, será então considerado o valor 0 (zero). A tabela abaixo traz exemplos destes literais numéricos:

Literais numéricos válidos	• -1.34E-2 • 1.0e+6 • 1.234E6 • 1_254.4879
Literais numéricos inválidos	• 1.34E-2.10 • e +6 • _254.4879

Tabela 4-4. Literais numéricos com base decimal

Constantes Booleanas

Para constantes booleanos podem ser usadas as palavras reservadas TRUE ou FALSE; ou seus equivalentes numéricos 1 ou 0.

Tipos de Dados

O tipo de dados define a forma que os dados podem ser armazenados na memória do CP. Esta seção define os tipos de dados possíveis e também as funções de conversão aplicáveis de um tipo para outro.

Tipos de Dados Básicos

A linguagem ST utiliza os tipos definidos pela tabela a seguir:

Tipo	Descrição	Bytes	Faixa
BOOL	Booleano	1	FALSE, TRUE, 1 ou 0.
BYTE	Seqüência de 8 bits	1	Nota 1
WORD	Seqüência de 16 bits	2	Nota 1
DWORD	Seqüência de 32 bits	4	Nota 1
USINT	Short integer	1	0 .. 255
INT	Integer	2	- 32.768 .. 32.767
DINT	Double integer	4	-2.147.483.648 .. 2.147.483.647
REAL	Real	4	$\pm 10^{\pm 38}$ Nota 2

Tabela 4-5. Tipos básicos de dados

Nota 1: Faixa numérica não é aplicável para os tipos BYTE, WORD e DWORD.
Nota 2: O ponto flutuante com precisão simples é definido pela norma IEC 559 (IEEE 754).

Classificação dos Dados

Os diversos tipos de dados são classificados em grupos e respeitando uma hierarquia. Mais adiante será visto que o uso de operadores normalmente estará limitado para determinados tipos de um mesmo grupo.

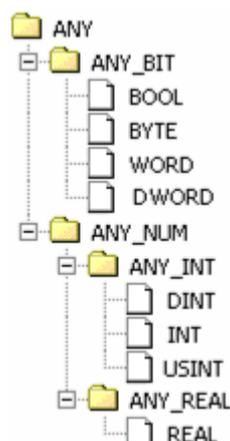



Figura 4-2. Classificação dos dados

Conversão de Tipos

As variáveis podem ser convertidas para outro formato através das funções de conversão de tipo. As conversões de um tipo para outro podem ocorrer de maneira implícita, onde o verificador insere a função de conversão automaticamente; ou de maneira explícita, onde o usuário deve inserir a função de conversão.

As funções de conversão de tipo utilizam o formato:

```
<tipo origem>_TO_<tipo destino>( <valor para converter> )
```

As conversões implícitas entre tipos são possíveis desde que não ocorra risco de perda de informação e que haja uma compatibilidade entre os tipos. A figura, a seguir, apresenta todas as conversões implícitas possíveis entre dois tipos:

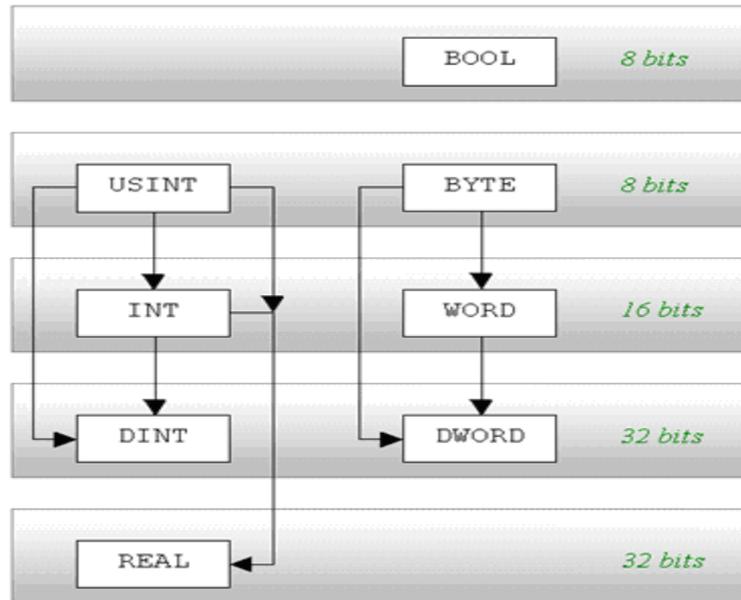



Figura 4-3. Conversões implícitas

A tabela, a seguir, demonstra as operações realizadas na conversão de tipos. Observa-se que mesmo as conversões implícitas possuem funções de conversão para o caso de conversão antes de uma operação matemática cujo resultado requeira maior resolução numérica.

Origem	Destino	Comentário	Conversão	Função
BOOL	BYTE	Se origem for TRUE o destino será igual a 1. Se origem igual à FALSE o destino será igual a 0.	Explícita	BOOL_TO_BYTE
	WORD		Explícita	BOOL_TO_WORD
	DWORD		Explícita	BOOL_TO_DWORD
	USINT		Explícita	BOOL_TO_USINT
	INT		Explícita	BOOL_TO_INT
	DINT		Explícita	BOOL_TO_DINT
	REAL		Explícita	BOOL_TO_REAL
BYTE	BOOL	Se a origem for igual a zero o destino será FALSE, caso contrário será TRUE.	Explícita	BYTE_TO_BOOL
	USINT	O destino receberá o valor na faixa de 0 a 255.	Explícita	BYTE_TO_USINT
	WORD	Converte o valor absoluto da origem para formato do destino. Os valores permanecerão inalterados.	Implícita	BYTE_TO_WORD
	DWORD		Implícita	BYTE_TO_DWORD
	INT		Explícita	BYTE_TO_INT
	DINT		Explícita	BYTE_TO_DINT
	REAL		Explícita	BYTE_TO_REAL
WORD	BOOL	Se a origem for igual a zero o destino será FALSE, caso contrário será TRUE.	Explícita	WORD_TO_BOOL
	BYTE	O destino receberá o byte menos significativo da origem.	Explícita	WORD_TO_BYTE
	USINT		Explícita	WORD_TO_USINT
	INT	O destino receberá o valor na faixa de –32768 a 32767, podendo assumir valores negativos.	Explícita	WORD_TO_INT
	DWORD	Converte o valor absoluto da origem para formato do destino. Os valores permanecerão os mesmos.	Implícita	WORD_TO_DWORD
	DINT		Explícita	WORD_TO_DINT
	REAL		Explícita	WORD_TO_REAL
DWORD	BOOL	Se a origem for igual a zero o destino será FALSE, caso contrário será TRUE.	Explícita	DWORD_TO_BOOL
	BYTE	O destino receberá o byte menos significativo da origem.	Explícita	DWORD_TO_BYTE
	USINT		Explícita	DWORD_TO_USINT
	WORD	O destino receberá o word menos significativo da origem	Explícita	DWORD_TO_WORD
	INT	O destino receberá a forma binária da origem, podendo assumir valores negativos.	Explícita	DWORD_TO_INT
	DINT		Explícita	DWORD_TO_DINT
	REAL	A conversão para REAL ocorre com perda de resolução numérica (apenas para valores com módulo maior que 16777215)	Explícita	DWORD_TO_REAL
USINT	BOOL	Se a origem for igual a zero o destino será FALSE, caso contrário será TRUE.	Explícita	USINT_TO_BOOL
	BYTE	O destino recebe a forma binária da origem.	Explícita	USINT_TO_BYTE
	WORD		Explícita	USINT_TO_WORD
	DWORD		Explícita	USINT_TO_DWORD
	INT	Converte o valor absoluto da origem para formato do destino. Os valores permanecerão os mesmos.	Implícita	USINT_TO_INT
	DINT		Implícita	USINT_TO_DINT
	REAL		Explícita	USINT_TO_REAL
INT	BOOL	Se a origem for igual a zero o destino será FALSE, caso contrário será TRUE.	Explícita	INT_TO_BOOL
	BYTE	O destino receberá o byte menos significativo da origem.	Explícita	INT_TO_BYTE
	USINT		Explícita	INT_TO_USINT

Origem	Destino	Comentário	Conversão	Função
	WORD	O destino recebe a forma binária da origem.	Explícita	INT_TO_WORD
	DWORD	O destino recebe a forma binária da origem. Caso a origem seja um número negativo o word mais significativo receberá o valor 0xFFFF.	Explícita	INT_TO_DWORD
	DINT	Converte o valor absoluto da origem para formato do destino. Os valores permanecerão os mesmos.	Implícita	INT_TO_DINT
	REAL		Implícita	INT_TO_REAL
DINT	BOOL	Se a origem for igual a zero o destino será FALSE, caso contrário será TRUE.	Explícita	DINT_TO_BOOL
	BYTE	O destino receberá o byte menos significativo da origem.	Explícita	DINT_TO_BYTE
	USINT		Explícita	DINT_TO_USINT
	WORD	O destino recebe a forma binária do word menos significativo da origem.	Explícita	DINT_TO_WORD
	INT		Explícita	DINT_TO_INT
	DWORD	O destino recebe a forma binária da origem.	Explícita	DINT_TO_DWORD
	REAL	O destino receberá o valor inteiro da origem. A conversão para REAL ocorre com perda de resolução numérica para valores com módulo maior que 16777215.	Explícita	DINT_TO_REAL
REAL	BOOL	Se a origem for igual a zero o destino será FALSE, caso contrário será TRUE.	Explícita	REAL_TO_BOOL
	WORD	A origem passa por uma conversão intermediária para DINT. Vide conversão de DINT para os demais tipos.	Explícita	REAL _TO_WORD
	INT	Converte o valor absoluto da origem para o destino. Valores fora da faixa numérica serão saturados nos limites numéricos do INT (-32768 a 32767).	Explícita	REAL _TO_INT
	DINT	O destino receberá o valor inteiro da origem. Valores fora da faixa numérica serão saturados nos limites numéricos do DINT (-2147483648 a 2147483647).	Explícita	REAL _TO_DINT

Tabela 4-6. Operações realizadas nas conversões de tipos

Paralelamente a essas operações de conversões ainda existem funções capazes de converter qualquer tipo de entrada para um tipo especifico. Estas funções de conversão de tipo utilizam o formato:

```
ANY_TO_<tipo destino>( <valor para converter> )
```

ATENÇÃO:
A conversão de varáveis do tipo REAL para qualquer variável do tipo ANY_INT utilizando as funções de conversão de tipo, sempre será passado a parte inteira da variável REAL. Estas funções não realizam arredondamento.

Variáveis

Uma variável é uma área de memória que armazena um tipo de dado da linguagem, estes dados são definidos como Tipos de Dados Básicos. Todas as variáveis devem ser declaradas antes de serem utilizadas.

Declaração de Variáveis

Toda declaração de variáveis deve ser feita entre as palavras reservadas VAR e END_VAR. Cada declaração pode conter diversas variáveis, separadas por vírgula. Neste caso, todas serão de um mesmo tipo.

```
VAR
<Var1>, <Var2>, ... , <VarM> : <tipo1>;
       <VarN>, <VarO>, ... , <VarZ> : <tipoN>;
END_VAR
```

As variáveis são sempre iniciadas automaticamente com o valor padrão, conforme a tabela a seguir. Somente as variáveis mapeadas em operandos do CP não são iniciadas automaticamente.

Tipo de dado	Valor inicial padrão
BOOL	FALSE ou 0
USINT, INT, DINT	0
BYTE, WORD, DWORD	0
REAL	0.0

Tabela 4-7. Valor de inicial das variáveis

Variáveis Somente-Leitura

Uma variável declarada como somente leitura só aceita atribuição de valor na sua declaração. Qualquer outra atribuição durante o código irá gerar erro em tempo de verificação.

A declaração de variáveis somente-leitura, também conhecidas como variáveis constantes, são feitas através da cláusula CONSTANT, conforme é mostrado abaixo:

```
VAR CONSTANT
       <Var1>, <Var2>, ... , <VarM> : <tipo1>;
       <VarN>, <VarP>, ... , <VarZ> : <tipoN>;
END_VAR
```

A aplicação destas variáveis ocorre quando se torna necessário mapear uma variável do CP dentro do módulo ST e que não possa sofrer operações de escrita dentro destes. Outra aplicação ocorre quando se deseja substituir constantes numéricas por símbolos e concentrar em um só lugar os valores associados a eles.

O exemplo, a seguir, mostra uma simples aplicação de variáveis do tipo CONSTANT:

```
VAR CONSTANT
       INI : INT := 1;
       FIM : INT := 100;
END_VAR

VAR
       VETOR                 : ARRAY[1..100] OF INT;
       NIVEL AT %F0030       : REAL;
       I                     : INT;
END_VAR

FOR I := INI TO FIM DO
       VETOR[ I ] := REAL_TO_INT( NIVEL * 100);
END_FOR;
```

Declaração de Vetores

Vetores são declarados usando-se a cláusula ARRAY. Os limites inferiores e superiores podem ser livremente especificados. Isto é, é possível a definição de um vetor de 10 elementos, sendo que o primeiro elemento é acessado pelo índice 14 e o último por 23, por exemplo. Estes vetores podem ser indexados através de qualquer expressão do tipo USINT, INT ou DINT, desde que seja positivo. O número máximo de elementos está limitado apenas pelo tamanho da área de dados (3 Kbytes).

```
VAR
       <vetor1> : ARRAY [ <limite_inferior> .. <limite_superior> ] OF <tipo>;
       <vetor2> : ARRAY [ <limite_inferior> .. <limite_superior> ] OF <tipo>;

END_VAR
```

Uma declaração de vetor onde o limite superior seja menor que o limite inferior ou onde o limite inferior seja menor que zero irá causar um erro de verificação.

Iniciando as Variáveis

Variáveis podem ser iniciadas com um valor diferente do padrão colocando este valor depois do tipo como é mostrado a seguir:

```
VAR
       <nome da variável> : <tipo> := <valor inicial>;
END_VAR
```

Um vetor também pode ser iniciado na declaração. Neste caso, os valores são escritos em seqüência e devem ser separados por vírgula.

```
VAR
       Vetor : ARRAY [1..10] OF INT := [ 11, 12, 13, 0, 0, 0, 0, 18, 19, 20 ];
END_VAR
```

Isto equivale a:

```
Vetor[ 1 ] := 11;
Vetor[ 2 ] := 12;
Vetor[ 3 ] := 13;
Vetor[ 4 ] := 0;
Vetor[ 5 ] := 0;
Vetor[ 6 ] := 0;
Vetor[ 7 ] := 0;
Vetor[ 8 ] := 18;
Vetor[ 9 ] := 19;
Vetor[ 10 ] := 20;
```

Mapeando Variáveis em Operandos Simples e Tabela

Todas as variáveis são alocadas pelo verificador em uma área de dados reservada para o uso de módulos P e F. Como esta área de memória é destruída no final da chamada do módulo todos os valores das variáveis também são.

O mapeamento de variáveis em operandos %M permite manter o valor da variável entre as chamadas do módulo. Da mesma forma, permite acessar os operandos dentro do CP utilizados por outros módulos. É possível mapear operandos %E, %S, %A, %M, %F, %I, %TM, %TF e %TI.

Todo mapeamento é feito na declaração de variáveis com a cláusula **AT**. Por mapear um endereço global para todos os módulos de programação do CP, esta operação tem as seguintes restrições:

- Só poderá ser usado dentro de VAR.
- Variáveis declaradas com mapeamento de operandos não são automaticamente iniciadas com seu valor padrão, mas podem ser iniciadas explicitamente.
- O uso da palavra reservada CONSTANT será permitido, e serve para indicar que a variável não será modificada no decorrer do programa.
- É obrigatório o uso do sinal “%” antes do operando.

Abaixo segue a sintaxe da declaração:

```
VAR
       <variável> AT <operando_CP> : <tipo> ;
END_VAR
```

Os tipos permitidos na declaração devem ser compatíveis com os operandos. A tabela, a seguir, demonstra as possíveis associações entre operandos e tipos:

Operando	Tipos permitidos
%M.x, %A.x, %E.x e %S.x	BOOL
%A, %E e %S	USINT ou BYTE
%M ou %TM	INT ou WORD
%I ou %TI	DINT ou DWORD
%F ou %TF	REAL

Tabela 4-8. Associação de tipos com operandos do CP

Também é possível associar bit de operando para uma variável do tipo BOOL. Os operandos permitidos são: %E, %S, %A e %M. Neste caso o mapeamento é feito para um operando do tipo BOOL.

A seguir, são mostrados alguns exemplos de mapeamentos permitidos:

Mapeamentos para BOOL:

```
VAR
       PRESSAO_ALTA AT %M0006.6 : BOOL;
       DIPARA_ECR   AT %A0006.6 : BOOL := FALSE;
       ENTRADA      AT %E0010.5 : BOOL;
       SAIDA        AT %S0008.7 : BOOL;
END_VAR
```

Mapeamentos para INT e WORD:

```
VAR
       NIVEL           AT %M0123:     INT;
       TEMPERATURA     AT %TM010[15]: INT;
       ALARMES         AT %M1432:     WORD;
END_VAR
```

Todos os mapeamentos de operandos serão verificados durante a execução do módulo para que se tenha certeza de que este operando foi declarado no módulo C. Caso ele não exista, é gerado um erro em tempo de execução, conforme descrito na Tabela 5-2.

Mapeando Vetores em Operandos Simples e Tabela

Também é possível mapear vetores para operandos do CP utilizando conjuntamente as cláusulas AT e ARRAY. Podem ser mapeados blocos de operandos simples, operandos tabela inteiros ou partes de operandos tabela.

Como no mapeamento de variáveis, no mapeamento de vetores o tipo do vetor deve ser compatível com o tipo do operando conforme a Tabela 4-5.

A seguir são citados exemplos de diferentes mapeamentos de vetores em operandos:

```
VAR
       (*Declarando um vetor de 150 posições no bloco de operandos M1000 a M1149*)
       ALARMES AT %M1000 : ARRAY[ 1..150 ] OF WORD;

       (*Declarando um vetor de 100 posições no bloco de operandos
         %TM200[000] a %TM200[099]. É obrigatório fornecer a posição
         inicial da tabela. *)
       PRESSOES AT %TM200[000] : ARRAY[ 1..100 ] OF INT;

       (*Declarando um vetor de 50 posições no bloco de operandos
       %TM030[050] a %TM030[099]. *)
```

```
        SILOS AT %TM030[050] : ARRAY[ 1..50 ] OF INT;
END_VAR
```

Nos mapeamentos de operandos, todo o bloco deve estar declarado no módulo C do CP. Da mesma forma um vetor mapeado em operando tabela deve estar totalmente contido no operando tabela mapeado. Caso alguma destas condições seja falsa é gerado um erro em tempo de execução.

Observação: Vetores mapeados em operandos não podem ser do tipo BOOL.

Funções

Uma função é uma rotina que pode ser executada diversas vezes. A utilização principal das funções é permitir uma maior modularidade para um programa. Uma função é declarada da seguinte forma:

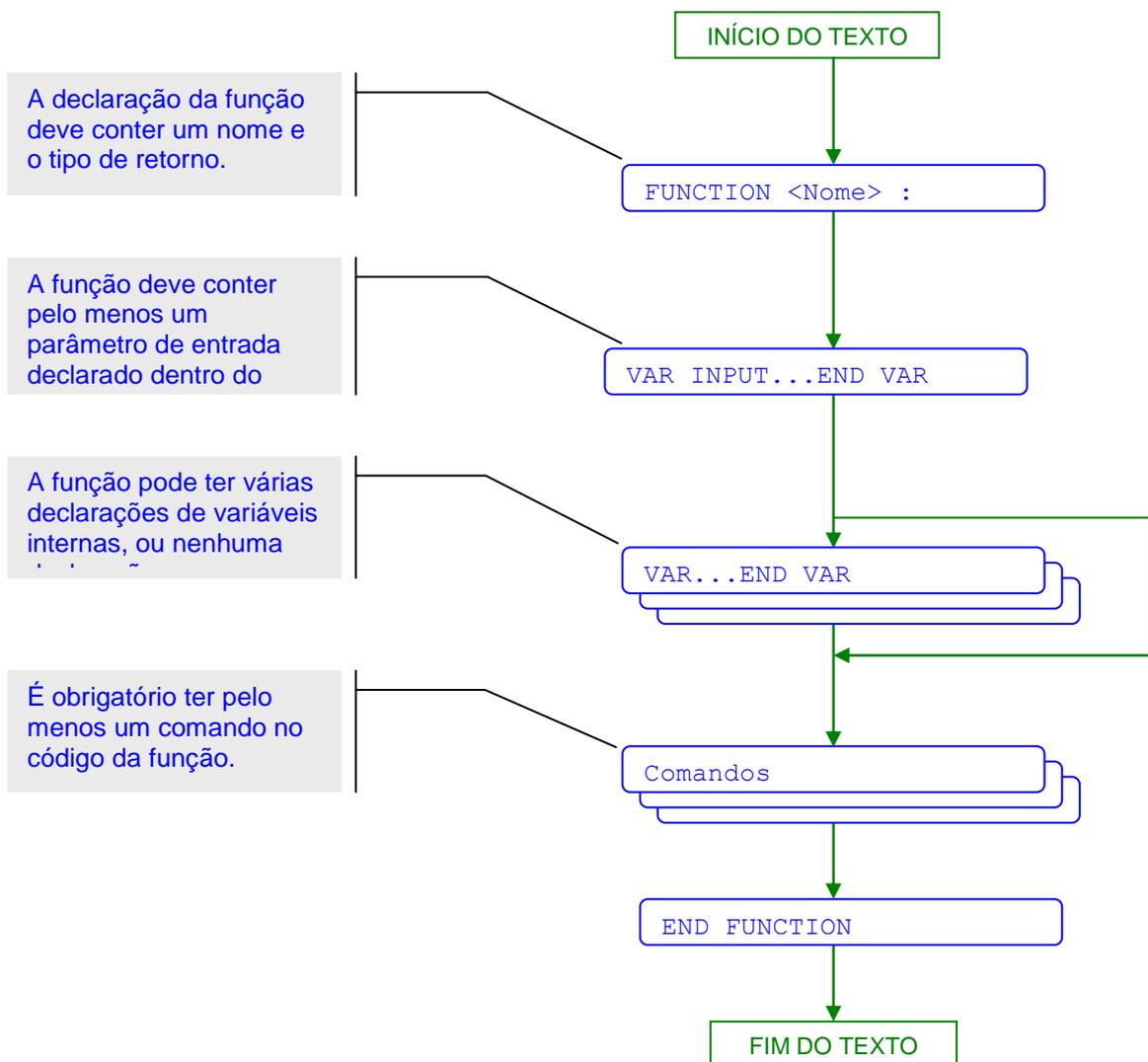



Figura 4-4. Estrutura do texto para definir uma função

Antes que a função retorne à rotina que a chamou, deve ser configurado um valor de retorno. Esta operação pode ser feita atribuindo-se um valor para o nome da função. O valor de retorno pode ser atribuído em diversas partes dentro do bloco de comandos e pode ser feito mais de uma vez.

```
FUNCTION FUNCAO1 : INT
        VAR_INPUT
                A : INT;
        END_VAR
```

```
        FUNCAO1 := A * 2;

END_FUNCTION
```

Os parâmetros de entrada devem ser especificados entre as palavras reservadas VAR_INPUT e END_VAR. A ordem que os parâmetros forem especificados determina a ordem em que estes devem ser passados para a função.

Os parâmetros de entrada não podem ser mapeados em operandos do CP ou serem definidos como vetores.

Uma função pode chamar outras funções definidas pelo usuário. Porém, a chamada da mesma função recursivamente não é permitida e gera um erro em tempo de verificação. O limite de chamadas aninhadas de funções é de 16 chamadas, ou seja, a partir do programa principal, somente podem ser chamadas 16 funções conforme mostrado na figura a seguir:

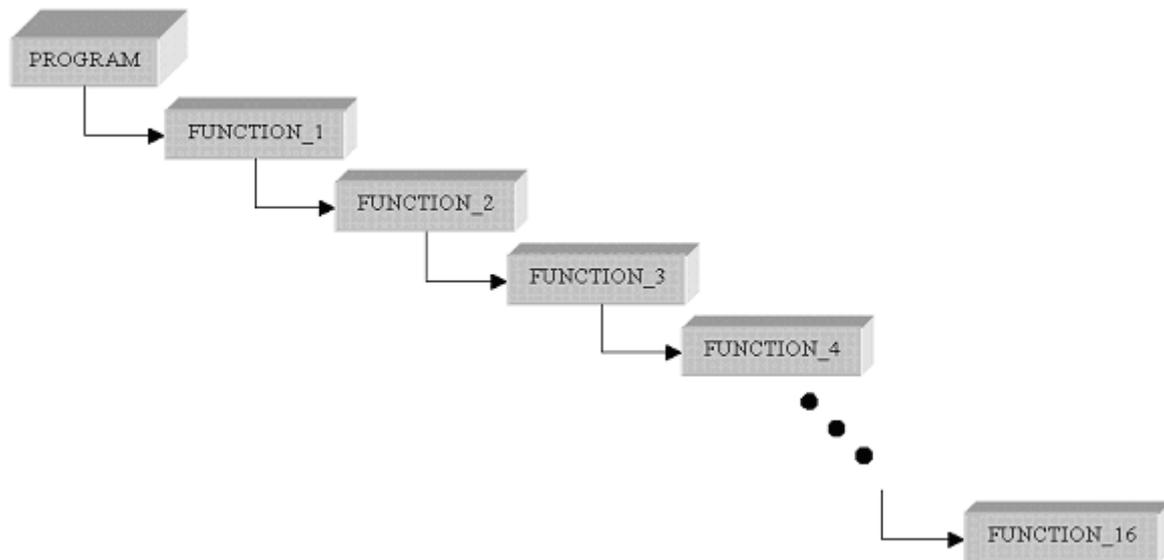



Figura 4-5. Limite de chamadas aninhadas de funções

Programa

O programa é a rotina de código onde começa a execução do módulo. Um programa pode acessar todas as variáveis globais e funções definidas no módulo. É declarado da seguinte forma:

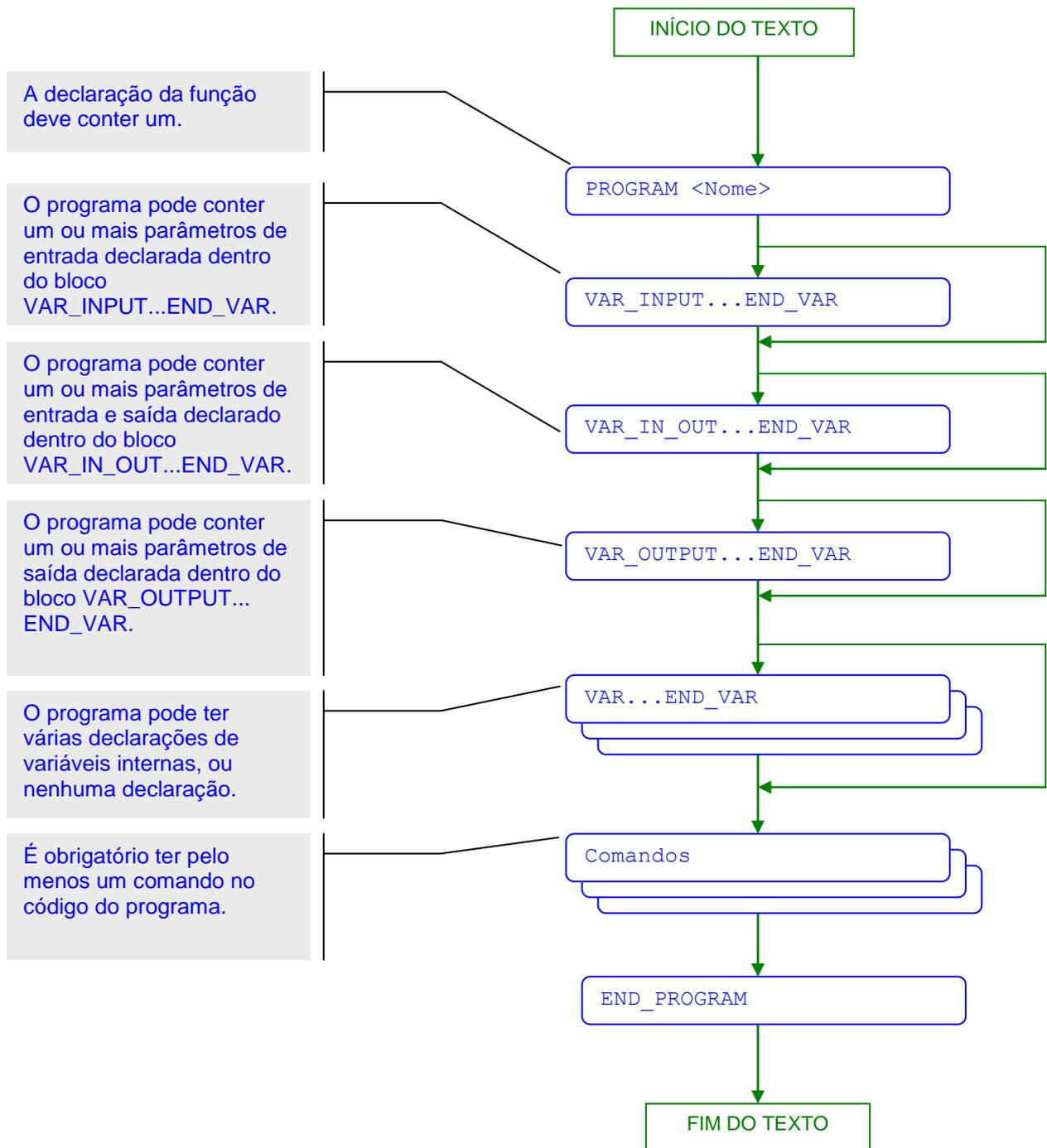



Figura 4-6. Estrutura do texto para definir uma função

Passagem de Parâmetros

Sempre que se estiver programando um módulo F, é possível a passagem de parâmetros de entrada e saída. Estes são passados pela instrução de ladder e estão limitadas as quantidades de parâmetros permitidas pela instrução, sendo um total de 10 parâmetros de entrada para entrada e/ou entrada-saída e mais 10 parâmetros de saída.

Declaração	Descrição	CHF - Chamada de módulo F
VAR_INPUT	Parâmetros de entrada. Podem apenas ser lidos.	Os primeiros n parâmetros de entrada.
VAR_IN_OUT	Parâmetros de entrada e saída. Podem ser lidos e escritos.	Os últimos n parâmetros de entrada.
VAR_OUTPUT	Parâmetros de saída. Podem ser apenas escritos.	Os parâmetros de saída.

Tabela 4-9. Tipos de parâmetros

A figura, a seguir, demonstra a associação entre as variáveis de entrada e saída de uma CHF com os tipos de variáveis utilizadas no módulo ST.

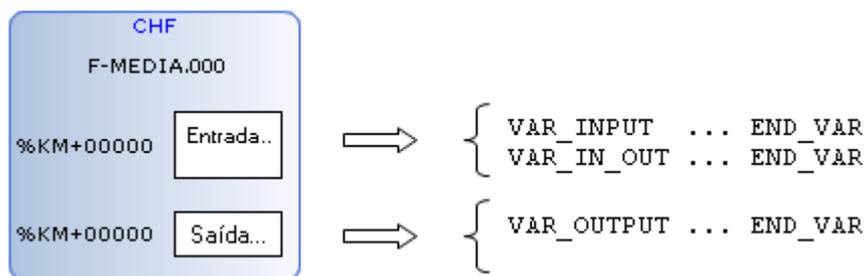



Figura 4-7. Relação entre entradas e saídas da CHF com operandos do módulo ST

Todos os parâmetros são passados por valor, podendo estes ser operandos tabela, blocos de operandos simples. As regras para consistência das passagens de parâmetros são mostradas a seguir e qualquer violação irá gerar um erro em tempo de execução:

- Os parâmetros devem ser passados para tipos equivalentes conforme a Tabela 4-8.
- Operandos constantes, %KM, %KI e %KF, só podem estar relacionados à VAR_INPUT e somente para variáveis simples, não podendo ser vetor.
- Não pode ser passado subdivisão de operandos como bit, nible, byte ou word.
- Operandos tabela podem estar relacionados somente a vetor.
- Um vetor deve estar relacionado a um operando tabela ou a um bloco de operandos simples, onde a primeira posição do vetor é o operando passado na CHF.
- Todos os operandos passados por parâmetros devem estar declarados.
- Todos os operandos relacionados a vetor devem estar declarados, sendo que isto inclui o tamanho completo do vetor.

A Tabela 4-9 representa a declaração dos operandos de entrada e saída para a seguinte declaração de variáveis em um módulo ST.

```
(*Lembrando que este exemplo é possível somente de ser executado em módulos F, pois este permite variáveis de entrada. Se for executado em módulos P apresentará erros de verificação.*)

PROGRAM TESTE
VAR_INPUT
        OP_INPUT0:          ARRAY[ 1..100 ] OF INT;
        OP_INPUT1:          INT;
        OP_INPUT2:          INT;
END_VAR

VAR_IN_OUT
        OP_IN_OUT0:         REAL;
        OP_IN_OUT1:         REAL;
        OP_IN_OUT2:         REAL;
END_VAR

VAR_OUTPUT
        OP_OUTPUT0:         ARRAY[ 1..10 ] OF INT;
        OP_OUTPUT1:         INT;
        OP_OUTPUT2:         INT;
        OP_OUTPUT3:         REAL;
        OP_OUTPUT4:         REAL;
        OP_OUTPUT5:         REAL;
```

```
END_VAR
;
END_PROGRAM
```

Passagem de Parâmetros para um Módulo F

Um módulo F programado em linguagem ST possibilita a passagem de parâmetros dos tipos VAR_INPUT, VAR_IN_OUT e VAR_OUT. As variáveis do tipo VAR_INPUT e VAR_IN_OUT são declaradas no campo “Entrada...” enquanto que as variáveis do tipo VAR_OUT são declaradas no campo “Saída...”, da CHF. A distinção entre variáveis de entrada e variáveis de entrada e saída é feita no momento da declaração das variáveis no módulo ST. O exemplo abaixo descreve a declaração das variáveis de um módulo F com três variáveis de entrada, duas variáveis de entrada e saída e duas variáveis de saída. Neste caso, devem ser declaradas na CHF cinco variáveis no campo “Entrada” e duas variáveis no campo “Saída”.

```
PROGRAM <nome do programa>
VAR
        VAR_0:  INT;
        VAR_1:  INT;
        VAR_2:  INT;
END_VAR

VAR_IN_OUT
        VAR_IN_OUT_0:       REAL;
        VAR_IN_OUT_1:       REAL;
END_VAR

VAR_OUTPUT
        VAR_OUT_0:          INT;
        VAR_OUT_1:          INT;
END_VAR

        <corpo do programa>

END_PROGRAM
```

No final da execução do módulo ST os operandos do tipo VAR_IN_OUT são copiados para seus respectivos operandos de origem.

ATENÇÃO: Somente parâmetros do tipo VAR_INPUT podem ser associados à operandos constantes (%KM, %KI, %KF) através dos parâmetros de entrada da CHF.

Sinais de Entrada e Saída do Módulo

Dentro do escopo de PROGRAM, existem até seis variáveis do tipo BOOL pré-declaradas referentes aos sinais de entrada e saída das instruções CHF e CHP. A figura, a seguir, apresenta esta associação:

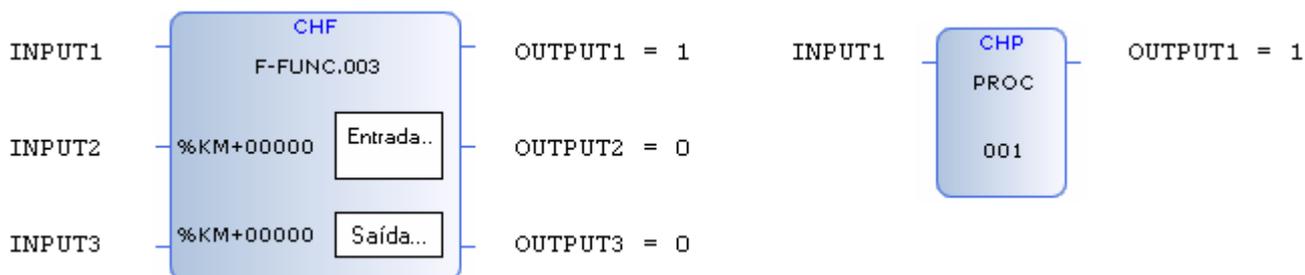



Figura 4-8. Sinais de entrada e saída dos módulos

As variáveis INPUT1, INPUT2 e INPUT3 são somente leitura. O valor de cada variável corresponde ao valor da entrada da instrução CHF ou CHP no ladder.

ATENÇÃO:
Para o programa ser executado é necessário que a primeira entrada da instrução CHF e CHP esteja habilitada.

Já as variáveis OUTPUT1, OUTPUT2 e OUTPUT3 são somente à escrita. A variável OUPUT1 é iniciada com TRUE enquanto as demais são iniciadas com FALSE. Caso ocorra um erro em tempo de execução, a variável OUTPUT1 é colocada em FALSE, independente do que o usuário já tenha escrito nela.

```
PROGRAM SOMADOR
VAR_INPUT
        I ,J : INT;
END_VAR
VAR
        Soma : INT;
END_VAR
        (* Entrada 2 da função define se a soma deve ser duplicada ou quadruplicada *)
        IF INPUT2 THEN
                Soma := (I + J) * 2;
        ELSE
                Soma := (I + J) * 4;
        END_IF;
        (* A saída da função será desenergizada se houver um estouro nas operações
           acima *)
        IF INTERNAL_OVERFLOW THEN
                OUTPUT1 := FALSE;
        ELSE
                OUTPUT1 := TRUE;
        END_IF;

END_PROGRAM
```

Variável Interna de Controle

INTERNAL_OVERFLOW

Indica que ocorreu um estouro positivo ou negativo da na última operação aritmética do tipo soma, subtração, multiplicação, negação ou da função EXPT.

```
PROGRAM SOMADOR

VAR_INPUT
        I ,J, K : INT;
END_VAR

VAR
        TEMP ,SOMA : INT;
END_VAR

        TEMP := I + J;
        IF INTERNAL_OVERFLOW THEN
                OUTPUT1 := FALSE;
                RETURN;
        END_IF;

        SOMA := TEMP + K;

        IF INTERNAL_OVERFLOW THEN
                OUTPUT1 := FALSE;
        END_IF;

END_PROGRAM
```

Quando ocorrer um estouro o resultado da operação estará limitado aos limites do tipo utilizado na operação. Por exemplo, somar duas variáveis do tipo INT cujos valores sejam 15_000 e 25_000, o resultado será 32_767.

Regras de Escopo e Tempo de Vida

Os nomes utilizados para identificar as variáveis, funções e o programa podem ser declarados em escopo global ou local.

No escopo global, o nome é visível a todas as funções e ao programa. Todas as funções e o programa são declarados no escopo global, já as variáveis, somente são globais aquelas variáveis declaradas fora das funções e do programa, no início do código.

O escopo local é o escopo dentro das funções e do programa. As variáveis declaradas dentro deste escopo são visíveis apenas às funções em que foram declaradas.

Não é permitido declarar duas vezes o mesmo nome dentro do mesmo escopo. Contudo, quando o nome de uma variável local coincidir com um nome global será sempre utilizado o nome declarado no escopo global.

O tempo de vida das variáveis dependerá do local em que foram declaradas. Para variáveis dentro de funções, os valores destas são variáveis são destruídos no final da chamada da função. Os valores das variáveis declaradas no escopo global são destruídos no final da chamada do módulo, quando o programa retorna para o ladder. Exceção são as variáveis mapeadas em operandos que mantém o seu valor entre as chamadas do módulo.

Comandos

Um programa escrito em ST é composto por uma seqüência de “comandos”. Os tipos de comandos são:

- Comandos de atribuição
- Comando de chamada de função
- Comandos de controle de programa
- Comandos de seleção
- Comandos de repetição

Além dos comandos, o verificador da Linguagem ST é capaz de avaliar expressões matemáticas para o cálculo de valores.

Expressões

Expressões são usadas para calcular ou avaliar valores. Uma expressão é composta por diversos operandos e operadores, sendo que estes podem ser variáveis, literais ou chamadas de funções.

Os operadores podem utilizar um ou dois operandos. Quando utilizam apenas um operador são chamados de unários. Neste caso, sempre se localizam antes do operando. Quando utilizam dois operandos, são chamados de binários. Neste caso o operador deverá estar entre os operandos.

Os dois operadores usados em operações binárias, na maioria das operações, devem ser do mesmo tipo.

Quando forem utilizados operadores de tipos diferentes, deverá ser utilizada uma função de conversão, conforme descrito na seção **Conversão de Tipos.**

Operadores Matemáticos

Estes operadores realizam operações matemáticas entre dois operandos. Os operandos podem ser qualquer ANY_NUM, mas não podem ser diferentes entre si. O operador matemático sempre retorna o mesmo tipo dos operandos usados.

Operador	Descrição	Aplicação	Muda INTERNAL_OVERFLOW
+	Adição	ANY_NUM + ANY_NUM	Sim
-	Subtração	ANY_NUM – ANY_NUM	Sim
-	Negação (menos unário)	- REAL - DINT - INT	Sim
*	Multiplicação	ANY_NUM * ANY_NUM	Sim
/	Divisão	ANY_NUM / ANY_NUM	Não
MOD	Resto da divisão inteira	ANY_INT MOD ANY_INT	Não

Tabela 4-10. Operadores matemáticos básicos

A operação `Var1 MOD 0` irá retorna 0 (zero). Esta operação não irá gerar erro de divisão por zero.

Operadores Relacionais

Os operadores relacionais executam uma comparação entre dois tipos numéricos conforme descrito na Tabela 4-11. Os operandos devem ser do mesmo tipo e a operação retorna sempre em um tipo BOOL.

Operador	Descrição	Aplicação
<	Menos que	ANY_NUM < ANY_NUM
>	Mais que	ANY_NUM > ANY_NUM
<=	Menor ou igual	ANY_NUM <= ANY_NUM
>=	Maior ou igual	ANY_NUM >= ANY_NUM
=	Igual	ANY = ANY
<>	Diferente	ANY <> ANY

Tabela 4-11. Operadores relacionais

Operadores Lógicos e Bit-a-Bbit

Estes operadores executam duas operações diferentes: lógica booleana e lógica bit-a-bit. A seleção da operação é feita de acordo com os tipos dos operandos usados.

Operações de lógica booleana são executadas entre operandos do tipo BOOL. A Tabela 4-12 representa o resulta de uma operação booleana. O resultado sempre será do tipo BOOL.

Operando A	Operando B	AND, &	OR	XOR
FALSE	FALSE	FALSE	FALSE	FALSE
FALSE	TRUE	FALSE	TRUE	TRUE
TRUE	FALSE	FALSE	TRUE	TRUE
TRUE	TRUE	TRUE	TRUE	FALSE

Tabela 4-12. Operadores lógicos

Operações lógicas bit-a-bit são executadas quando os operandos são BYTE, WORD e DWORD, sendo que os dois operandos devem ser do mesmo tipo. A operação de bit-a-bit realiza uma operação booleana para cada bit dos operandos, conforme descrito na Tabela 4-13. Estas operações retornam o mesmo tipo dos operandos usados.

```
PROGRAM SOMADOR
VAR
       Entradas  : BYTE ;
       Filtros   : BYTE ;
       Alarmes   : BYTE ;

END_VAR
       Entradas := 2#0011_1001;         (* Entradas, do tipo BYTE *)

       Filtros  := 2#0001_1100;         (* Filtros,  do tipo BYTE *)

       Alarmes := Entradas AND Filtros; (* Alarmes,  do tipo BYTE *)
```

```
                                  (*Alarmes é igual a 2#0001_1000 *)
END_PROGRAM
```

Operador	Descrição	Aplicação
AND, &	Operação "E"	ANY_BIT **AND** ANY_BIT ANY_BIT **&** ANY_BIT
XOR	Operação "OU" exclusivo booleano	ANY_BIT **XOR** ANY_BIT
OR	Operação "OU" booleano	ANY_BIT **OR** ANY_BIT
NOT	Complemento booleano	**NOT** ANY_BIT

Tabela 4-13. Operadores bit-a-bit

Precedência de Operadores

A avaliação da expressão é feita de acordo com a precedência dos operadores, conforme mostrado na Tabela 4-14. Operadores de maior precedência são avaliados primeiro. Caso os operadores tenham a mesma precedência, o que estiver mais a esquerda será o primeiro a ser avaliado.

Precedência	Operador	Descrição
0 (maior)	(...)	Expressão entre parênteses
	função (...)	Avaliação de função
2	-	Negação
	NOT	Complemento
3	*	Multiplicação
	/	Divisão
	MOD	Resto
4	+	Adição
	-	Subtração
5	< , > , <= , >=	Comparação
6	=	Igualdade
	<>	Desigualdade
7	AND, &	Operação "E" booleana
8	XOR	Operação "OU" exclusivo booleana
9 (menor)	OR	Operação "OU" booleana

Tabela 4-14. Precedência de operações

Chamadas de Função

Uma função pode ser chamada dentro de uma expressão. O valor a ser passado para cada parâmetro é escrito dentro dos parênteses e cada parâmetro é separado por vírgula. A ordem que os valores devem ser escritos deve ser a mesma ordem que os parâmetros foram declarados na função.

```
(* Chamada de função: primeira forma *)
nome_funcao ( expressao_1, expressao_2, ... , expressao_n )
```

Em uma expressão que tenha mais de uma função, não é possível determinar qual função será executada primeira.

Função de Potência

A função de potência retorna o resultado da operação ($base^{expoente}$) , onde base pode ser ANY_INT ou REAL e expoente pode ser USINT. O tipo do resultado será o mesmo tipo da base. Esta função altera o valor de INTERNAL_OVERFLOW.

```
PROGRAM XXXXXX
VAR
        base, resultado         : INT;
        expoente                : USINT;
END_VAR

base            :=      4;
expoente        :=      2;
```

```
resultado      :=      EXPT( base, expoente );

END_PROGRAM (* resultado é igual a 16*)
```

Constantes de Inteiros

As constantes de inteiros podem ser usadas em operações com ANY_INT. Desde que o valor do literal não ultrapasse o limite do tipo do outro operando.

Faixa	Tipos compatíveis
0 a 255	USINT, INT, DINT, BYTE, WORD e DWORD
–32.768 a -1	INT e DINT
0 a 32.767	INT, DINT, WORD e DWORD
0 a 65.535	DINT, WORD e DWORD
-2.147.483.648 a 1	DINT
0 a 2.147.483.647	DINT e DWORD
0 a 4.294.967.296	DWORD

Tabela 4-15. Constantes inteiras

Os literais numéricos reais só podem ser utilizados em operações com variáveis também do tipo REAL.

Comando de Atribuição

A atribuição é usada para escrever um determinado valor em uma variável.

```
<variável> := <expressão>;
```

Comando de Controle de Programa

Comando RETURN

Uma função sempre retorna para a rotina que a chamou após a execução da última afirmação. Porém, é possível retornar no meio do código, através do uso da palavra reservada **RETURN.**

No caso da palavra reservada **RETURN** ser usada no programa principal, o mesmo será interrompido, retornando o controle da execução do programa para o ladder.

Comandos de seleção

Um comando de seleção executa um dentre vários blocos de afirmações. A escolha do bloco é definida por uma função de avaliação expressa pelo não terminal *<expressão_booleana>*. Existem dois tipos de afirmações de seleção: o comando **IF** e o comando **CASE.**

Comando IF

O comando IF executa as afirmações após o **THEN** se a *<expressão_booleana>* de teste for verdadeira. Opcionalmente é possível inserir outras condições de teste com a clausula **ELSIF**, sendo que, apenas o grupo de afirmações onde o primeiro teste for verdadeiro é que será executado.

Opcionalmente, é possível especificar um bloco de afirmações para ser executado, caso todos os testes falhem, através da clausula **ELSE**.

```
IF <expressão_booleana> THEN <comandos>

{ ELSEIF <expressão_booleana>  THEN <comandos> }

[ ELSE <comandos> ]

END_IF;
```

Exemplo:

```
PROGRAM XXXXXX

VAR
        TESTE : INT;
        A, B, C, D, E: INT;

END_VAR

IF A = B THEN
        TESTE := 10;

ELSEIF A = C THEN
        TESTE := 11;

ELSEIF A = D THEN
        TESTE := 12;

ELSEIF A = E THEN
        TESTE := 13;

ELSE
        TESTE := 0;
END_IF;
END_PROGRAM
```

Comando CASE

O comando **CASE** também executa apenas um bloco de afirmações. A seleção do bloco é feita pela comparação do valor inteiro de *<expressão_inteira>* com os valores escritos nos *<casos>*.

Opcionalmente é possível especificar um bloco de afirmações para ser executado caso todos os testes falhem através da clausula **ELSE**.

```
CASE <expressão_inteira> OF

<casos> :  <comandos>

{ <casos> :  <comandos> }

[ ELSE <comandos> ]

END_CASE;
```

<casos> indica lista de valores ou sub-faixa separados por vírgulas.

Exemplo:

```
CASE Temperatura OF

        0 :  Bomba1 := 10;

        1, 2, 5..10 :  Bomba1 := 10;

END_CASE;
```

Os valores testados devem ser compatíveis com *<expressão_inteira>*.

Comandos de Repetição ou Iteração

Um comando de iteração executa repetidamente um bloco de afirmações. O número de vezes que é executado depende do tipo de iteração, que pode ser: o comando **WHILE**, o comando **FOR** e o comando **REPEAT**.

Para todos os comandos, é possível interromper o laço da iteração prematuramente através do comando **EXIT**. Este comando só pode ser usado dentro do laço da iteração. O uso do comando EXIT fora de uma afirmação de iteração causará um erro de verificação.

Comando WHILE

O comando WHILE executa um bloco de comandos enquanto a função de avaliação *<expressão_booleana>* for verdadeira. O comando WHILE sempre testa a função de avaliação antes

de executar o bloco. Assim, se na primeira iteração o teste resultar em falso, o bloco de afirmações não será executado.

```
WHILE <expressão_booleana> DO

       <comandos>

END_WHILE;
```

Exemplo:

```
PROGRAM XXXXXX
(* Declaração de variaveis*)
VAR
       i :                    INT ;    (* Variavel de controle  para o comando for *)
       Vetor :                ARRAY [1..10] OF INT ;

       INICIO_TABELA : INT;
       FIM_TABELA :   INT;
       ACUMULADOR :   INT;

END_VAR

INICIO_TABELA := 1;
FIM_TABELA := 10;

i := INICIO_TABELA;

WHILE I <= FIM_TABELA DO

       ACUMULADOR := ACUMULADOR + Vetor[ I ];
       I := I + 1;
END_WHILE;

END_PROGRAM
```

Comando REPEAT

O comando REPEAT executa o bloco de comandos até que função de avaliação *<expressão_booleana>* seja verdadeira. Diferente do comando WHILE, o comando REPEAT executa primeiro o bloco de afirmações e depois testa a função de avaliação. Assim o bloco de afirmações é executado pelo menos uma vez.

```
REPEAT

       <comandos>

UNTIL <expressão_booleana>  END_REPEAT;
```

Exemplo:

```
PROGRAM XXXXXX
(* Declaração de variaveis*)
VAR
       i :                    INT ;    (* Variavel de controle  para o comando for *)
       Vetor :                ARRAY [1..10] OF INT ;

       INICIO_TABELA : INT;
       FIM_TABELA :   INT;
       ACUMULADOR :   INT;

END_VAR

INICIO_TABELA := 1;
FIM_TABELA := 10;

i := INICIO_TABELA;

REPEAT
       ACUMULADOR := ACUMULADOR + Vetor[ I ];
       I := I + 1;
UNTIL I > FIM_TABELA  END_REPEAT;
```

```
END_PROGRAM
```

Comando FOR

O comando FOR permite executar um bloco de comandos repetidas vezes. O número de repetições é controlado por uma <*variável_controle*>. Esta variável deve ser do tipo USINT ou INT e não pode ser um operando da UCP do controlador programável (%M por exemplo).

Primeiramente, <*variável_controle*> é inicializada com o valor de <*expr_inicial*>. No inicio de cada repetição, é verificado se o valor de <*variável_controle*> excedeu o valor definido por <*expr_final*>. Se não excedeu, o bloco de comandos é executado. Caso contrário, o comando FOR é encerrado. No fim da execução do bloco, <*variável_controle*> é incrementada em 1, ou pelo valor definido por <*expr_inc*>. Tanto a <*variável_controle*> como as expressões <*expr_inicial*>, <*expr_inicial*> e <*expr_final*> devem ser dados do mesmo tipo (USINT ou INT).

```
FOR <variável_controle> := <expr_inicial> TO <expr_final> [ BY <expr_inc> ] DO

       <comandos>

END_FOR;
```

ou

```
FOR <variável_controle> := <expr_inicial> TO <expr_final> DO

       <comandos>

END_FOR;
```

A <*variável_controle*>, dentro do escopo do laço FOR, pode ser lida mas não pode ser escrita.

Durante as interações, será usado o valor de < *expr_final* > avaliado no inicio do FOR. Ou seja, esta expressão não é reavaliada no decorrer do comando.

Exemplo:

```
PROGRAM XXXXXX
(* Declaração de variaveis*)
VAR
i :                     INT ;    (* Variavel de controle  para o comando for *)
Vetor :                 ARRAY [1..10] OF INT ;

INICIO_TABELA : INT;
FIM_TABELA :   INT;
ACUMULADOR :   INT;

END_VAR

INICIO_TABELA := 1;
FIM_TABELA := 10;

FOR I := INICIO_TABELA TO FIM_TABELA DO

ACUMULADOR := ACUMULADOR + Vetor[ I ];

END_FOR;
END_PROGRAM
```

5. Depuração

Métodos de Depuração

Este capítulo descreve como deve ser efetuada a depuração de um módulo criado em linguagem ST, de acordo com as orientações do Capítulo 4 - Programação. Além do forçamento e monitoração de variáveis, existem outros recursos muito úteis quando se deseja depurar uma aplicação, seja ela programada em ladder ou linguagem ST.

Na seqüência são descritos dois métodos de depuração. O primeiro utiliza-se do recurso de execução em modo ciclado do controlador programável, enquanto que o segundo utiliza máquinas de estados para implementar a depuração.

Modo Ciclado

Quando em modo ciclado, o controlador programável não executa periodicamente o módulo E001, permanecendo à espera de comandos do programador MasterTool. Para maiores detalhes a respeito do modo ciclado deve ser consultado o Manual de Programação do MasterTool XE.

Utilizando-se a depuração em modo ciclado, pode-se, entre um ciclo e outro, efetuar a verificação dos valores das variáveis utilizadas e até mesmo se forçar valores para verificar o comportamento da aplicação que está sendo depurada.

Máquinas de estado

Este método de depuração consiste em criar uma seqüência definida de ações vinculadas a um índice ou estado. Desta forma, pode-se executar o código passo a passo, onde cada passo pode ser uma simples linha de código ou um trecho de código qualquer.

Uma implementação simples desta máquina de estados pode ser obtida através do comando IF, conforme é mostrado a seguir:

```
IF ESTADO = 1 THEN
< comandos bloco 1>
ESTADO := 0;
END_IF;

IF ESTADO = 2 THEN
< comandos bloco 2>
ESTADO := 0;
END_IF;
.
.
.
IF ESTADO = n THEN
< comandos bloco n>
ESTADO := 0;
END_IF;
```

Cada bloco de comandos do código apresentado é executado uma única vez, visto que o índice dos estados é zerado no final da execução do bloco. Incrementando-se a variável do estado manualmente, pode-se executar diferentes partes do código de forma controlada. Pode-se, entre um estado e outro, forçar diferentes valores para que se verifique o comportamento do código e os valores obtidos nas variáveis envolvidas, por exemplo.

Erros em Tempo de Verificação

Uma informação útil a nível de depuração dos códigos fontes são os erros retornados pelo verificador ao se verificar o código escrito. Esses erros apontam problemas de digitação, associações inválidas e utilização indevida de instruções, facilitando, assim, o processo de desenvolvimento do módulo ST.

O verificador da linguagem ST poderá gerar os seguintes erros de um módulo:

Descrição	**Causa provável**
Caractere inválido	O texto ou o caractere digitado não foi reconhecido pelo verificador.
Símbolo inválido	Indica que a notação léxica usada está incorreta. As notações possíveis estão listadas abaixo: Identificador: - Terminou com "_" - Possui dois "_" consecutivos. Literal numérico: - Terminou com "_" - Possui dois "_" consecutivos. - Ponto flutuante com expoente sem valor. - Digito inválido para base numérica.
Comentário não encerrado com *)	O fim de arquivo foi encontrado antes do fim do comentário.
Era esperado *<token 1>* ao invés de *<texto 2>*	Erro de sintaxe. Provavelmente foi digitado um texto errado, ou falta um texto.
Não era esperado *<texto 1>* após *<texto 2>*	Erro de sintaxe. Provavelmente foi digitado um texto errado, ou falta um texto..
Não era esperado *<texto 1>*	Erro de sintaxe. Provavelmente foi digitado um texto errado, ou falta um texto.
O identificador *<nome>* já foi declarado neste escopo.	O identificador já foi declarado. Utilize outro nome.
Variável *<nome>* não foi declarada.	A variável não foi declarada. Declare a variável antes de usá-la.
Array*<nome>* não foi declarado.	O array não foi declarado. Declare o array antes de usá-lo.
Função *<nome>* não foi declarada.	A função não foi declarada. Declare a função antes de usá-la.
Era esperada uma expressão inteira.	O comando CASE esperam inteiras para testá-las. A expressão fornecida não é inteira.
Impossível converter <tipo 1> para <tipo 2>	A conversão entre os tipos não é permitida. Tente utilizar uma função de conversão explicita.
Não é possível realizar a operação <operacao> entre os tipos <tipo à esquerda> e o tipo <tipo à direita>.	A operação não é válida para os tipos passados. Utilizar as funções de conversão de tipos para ajustar ao tipo correto.
Não é possível realizar a operação <operacao> com o <tipo à direita>.	A operação não é válida para o tipo passado. Utilizar as funções de conversão de tipos para ajustar ao tipo correto.
Comando EXIT não pode ser executado fora de um laço WHILE, FOR ou REPEAT.	Comando EXIT não pode ser executado fora de um laço WHILE, FOR ou REPEAT.
Este trecho de código nunca será executado.	O trecho de código foi escrito após um comando de RETURN ou EXIT e portanto nunca será executado.
Chamada recursiva da função <nome>	Uma função não pode ser chamada recursivamente.
Um dos caminhos da função <nome> não retorna valor.	Existe um caminho de código que não retorna valor através do comando: Função := valor;
Função *<nome>* foi chamada com parâmetros a mais que o declarado.	
Função *<nome>* foi chamada com parâmetros a menos que o declarado.	
Símbolo <nome> não é uma função	Era esperado que o símbolo fosse uma função.
Símbolo <nome> não é um vetor	Era esperado que o símbolo fosse um vetor.
Símbolo <nome> não é uma variável	Era esperado que o símbolo fosse uma variável.
Símbolo <nome> não é um constante	Era esperado que o símbolo fosse uma constante.

Símbolo <nome> não permite leitura	
Símbolo <nome> não permite escrita	
Constante <valor> já utilizada em outro caso.	A constante inteira fornecida em um caso já foi utilizada em outro caso do mesmo comando CASE.
Variável de controle do FOR <nome> não pode estar associada a operando do CP.	
Variável de controle do FOR <nome> não ser escrita dentro do FOR	
Número incorreto de elementos para iniciar o vetor	
Operando <tipo operando> do CP é incompatível com <tipo variável>	
<Tipo> não pode ser utilizado como parâmetro de <nome função ou programa>	O tipo declarado não pode ser utilizado como parâmetro. As funções não suportam vetores e programas não suporta o tipo BOOL e vetor de BOOL
O módulo F não pode ter mais de 10 parâmetros VAR_INPUT e VAR_IN_OUT	O número de parâmetros declarados em VAR_INPUT e VAR_IN_OUT não pode ser maior que 10 parâmetros.
O módulo F não pode ter mais de 10 parâmetros VAR_OUTPUT	O número de parâmetros declarados em VAR_OUTPUT não pode ser maior que 10 parâmetros.
Módulo procedimento não permite parâmetros	
FUNCTION não admite parâmetros do tipo VAR_IN_OUT ou VAR_OUTPUT	Foi declarada uma ou mais variáveis do tipo VAR_IN_OUT ou VAR_OUTPUT no escopo de uma FUNCTION.
Valor <valor> fora dos limites	
Valor mínimo maior que valor máximo	
O operando <operando> não é válido	O operando não foi corretamente digitado ou não é suportado pela linguagem ST.
Número de variáveis excede o limite	
UCP <nome> não é suportada pelo compilador ST	A UCP indicada não é suportada módulos ST pelo compilador ST. Não é possível gerar módulos ST para esta UCP.
Quantidade insufuciente de operandos <tipo do operando> temporários. Mínimo <valor> operandos.	Verifique se o operando temporário está habilitado e se a quantidade de operandos é maior ou igual ao valor indicado na mensagem. Atualmente é suficiente configurar apenas 04 operandos.
Número de elementos do array é inválido. Máximo de <quantidade> elementos.	
Falha na montagem do módulo	Houve uma falha na montagem do módulo. Entrar em contato com suporte da Altus.

Tabela 5-1. Erros em tempo de verificação

Erros em Tempo de Execução

Sempre que o módulo ST executar uma operação ilegal, como por exemplo uma divisão por zero ou mesmo um acesso a operandos não declarados, a execução do código é imediatamente interrompida e a execução do ladder passa para o módulo ladder com a instrução CHP ou CHF que chamou o módulo ST.

Para indicar o erro encontrado são utilizados os operandos de diagnósticos, conforme definidos na seção **Operandos de Diagnóstico**. Além de indicar o erro nos operandos, se houver um erro de tempo de execução a saída OUTPUT1 da instrução CHP/CHF é zerada.

São utilizados 5 operandos %M para indicar o motivo do erro, conforme a tabela a seguir:

Operando	Descrição
%M+0	Linha onde ocorreu o erro: Linha = -1: indica que um erro foi encontrado no início do módulo, antes da primeira linha de código. Linha = -32.768 ou bit15 em 1: nenhum erro foi encontrado, os demais operandos são zerados.
%M+1	Código do erro. Consulte a tabela a seguir.
%M+2	Primeira informação complementar ao erro.
%M+3	Segunda informação complementar ao erro.
%M+4	Terceira informação complementar ao erro.

Tabela 5-2. Erros em tempo de execução

A tabela, a seguir, apresenta uma descrição detalhada dos possíveis erros em tempo de execução:

Código	Descrição	Causa provável	Comp 1	Comp 2	Comp 3	Correção
2000	Acesso a operando simples não declarado.	O operando não foi declarado no módulo C.	Tipo do operando: 0: %M 8: %E/S 9: %A 4: %F 1: %I	Endereço do operando	Não usado	Declarar o operando no módulo C.
2001	Operando tabela não declarado	O operando tabela não foi declarado no módulo C.	Tipo do operando: 0: %M 4: %F 1: %I	Endereço da tabela	Não usado	Declarar o operando no módulo C.
2002	Posição da tabela não declarada	A tabela não foi declarada no módulo C com o número de posições utilizado pelo programa.	Tipo do operando: 0: %M 4: %F 1: %I	Endereço da tabela	Posição da tabela	Declarar o operando no módulo C.
2003	Parâmetro de entrada incorreto.	O operando passado por parâmetro de entrada na CHF não é compatível com o tipo declarado.	Número do parâmetro de entrada.	Não usado	Não usado	Corrigir o operando na instrução CHF.
2004	Parâmetro de saída incorreto.	O operando passado por parâmetro de saída na CHF não é compatível com o tipo declarado.	Número do parâmetro de saída.	Não usado	Não usado	Corrigir o operando na instrução CHF.
2005	Operando do parâmetro de entrada não declarado.	O operando passado por parâmetro de entrada na CHF não foi declarado no módulo C do CP. Ou os operandos do *vetor* não foram declarados no módulo C do CP.	Número do parâmetro de entrada.	Não usado	Não usado	Declarar o operando no módulo C ou corrigir o operando na instrução CHF.
2006	Operando do parâmetro de saída não declarado.	O operando passado por parâmetro de saída na CHF não é foi declarado no módulo C do CP. Ou os operandos do *vetor* não foram declarados no módulo C do CP.	Número do parâmetro de saída.	Não usado	Não usado	Declarar o operando no módulo C ou corrigir o operando na instrução CHF.
2008	Valor inválido para operando constante	O valor da constante utilizada excede o tamanho do parâmetro da CHF	Número do parâmetro de entrada ou de saída.	Não usado	Não usado	Corrigir o valor da constante utilizada.

Código	Descrição	Causa provável	Comp 1	Comp 2	Comp 3	Correção
2009	Número de parâmetros de entrada incorreto.	A quantidade de parâmetros de entrada declarada na CHF está incorreta.	Não usado	Não usado	Não usado	Verificar o número correto de parâmetros de entrada utilizados na instrução CHF.
2010	Número de parâmetros de saída incorreto.	A quantidade de parâmetros de saída declarada na CHF está incorreta.	Não usado	Não usado	Não usado	Verificar o número correto de parâmetros de saída utilizados na instrução CHF.
2011	Parâmetro de entrada do tipo array com quantidade insuficiente de posições.	Operandos do tipo tabela com número de posições insuficientes.	Número do parâmetro de entrada ou de saída.	Não usado	Não usado	Verificar o número correto de posições utilizadas no respectivo parâmetro de entrada.
2012	Parâmetro de saída do tipo array com quantidade insuficiente de posições.	Operandos do tipo tabela com número de posições insuficientes.	Número do parâmetro de saída.	Não usado	Não usado	Verificar o número correto de posições utilizadas no respectivo parâmetro de saída.
2015	Índice para vetor é invalido.	O índice fornecido para acessar um *vetor* é menor que seu limite inferior ou superior ao seu limite superior.	Não usado	Não usado	Não usado	Verificar os possíveis valores que podem ser passados como índice do vetor no programa.
2020	Divisão por zero.	Ocorreu uma divisão por zero.	Não usado	Não usado	Não usado	Verificar os possíveis valores que podem ser utilizados como divisor na operação.
2030	Tempo de execução excedido.	O tempo de execução permitido para o módulo foi excedido.	Não usado	Não usado	Não usado	Possivelmente uma instrução de laço como WHILE ou REPEAT foi executada infinitamente.
2031	Execução dentro da E018	O módulo esta sendo executado dentro da E018, o que é proibido.	Não usado	Não usado	Não usado	Não é possível chamar o módulo ST dentro da E018.
2032	Versão de executivo	A versão do executivo é menor que o esperado pela função.	Versão mínima do executivo (em decimal)	Não usado	Não usado	O módulo gerado em ST não pode ser executado em versões inferiores do executivo.
2040	Módulo salvo com erro	O módulo foi salvo com erros de verificação gerando um programa em branco.	Não usado	Não usado	Não usado	O módulo foi salvo e enviado ao CP com erros de verificação.
2050	Limite de chamadas aninhadas excedido	O número de chamadas de funções em seqüência excedeu o seu limite.	Não usado	Não usado	Não usado	Garantir que as chamadas aninhadas não excedam seu limite. Vide Limites de Software.

Tabela 5-3. Descrição dos erros em tempo de execução

6. Exemplos de Utilização

Este capítulo apresenta exemplos de programas escritos em ST.

Buffer de eventos

O módulo F-EVT.030 insere um evento em no buffer de eventos implementado no operando %TM0010. Cada evento ocupa 3 posições no buffer. A primeira posição armazena o valor do minuto no byte alto e de segundo no byte baixo. A segunda posição armazena a hora. Já a terceira posição armazena o código do evento.

```
(*
       Armazena eventos em uma TM. Cada evento é armazenado em 3 posições.
                      byte alto              byte baixo
       pos 0          minuto                 segundo
       pos 1                                 hora
       pos 2                                 evento

       O código de evento é passado por parâmetro da CHF.
*)

(* Variáveis globais ----------------------------------------------------------------- *)
(* Buffer de eventos *)
VAR
       BUFFER                AT  %TM0010[000]  :ARRAY[ 1..120 ] OF INT;(* Buffer eventos
*)
       BUFFER_IN             AT  %M0000   :INT;     (* Entrada no buffer *)
       BUFFER_OUT            AT  %M0001   :INT;     (* Saída do buffer *)
       BUFFER_NUM_EVENTOS    AT  %M0002   :INT;     (* Número de eventos armazenados *)
       BUFFER_OVERFLOW       AT  %A0001.0 :BOOL;    (* Indica overflow *)
END_VAR

(* Constantes *)
VAR CONSTANT
       BUFFER_INF            : INT := 1;       (* Primeiro índice do array *)
       BUFFER_SUP            : INT := 120;     (* Último índice do array *)
       BUFFER_LIMITE         : INT := 40;      (* Número máximo de elementos no buffer *)
END_VAR


(* Funções --------------------------------------------------------------------------- *)
(*
       Função: InserirValor

       Insere um valor na próxima posição do buffer de eventos;
       Retorna TRUE se ocorreu overflow;
 *)

FUNCTION INSERIR_VALOR : BOOL
VAR_INPUT
       VALOR : INT;
END_VAR

       (* Insere o valor na posição *)
       BUFFER[ BUFFER_IN ] := VALOR;

       (* Controla os limites do buffer *)
       IF BUFFER_IN = BUFFER_SUP     THEN
              BUFFER_IN := BUFFER_SUP;
       ELSE
              BUFFER_IN := BUFFER_IN + 1;
       END_IF;

       (* Controla o overflow *)
       IF BUFFER_NUM_EVENTOS = BUFFER_LIMITE THEN
              INSERIR_VALOR := TRUE;
       ELSE
              BUFFER_NUM_EVENTOS := BUFFER_NUM_EVENTOS + 1;
              INSERIR_VALOR := FALSE;
       END_IF;
```

```
END_FUNCTION


(* Programa ------------------------------------------------------------------------------ *)
PROGRAM F_EVT_030
        (* Parâmetros de entradas da CHF*)
        VAR_INPUT
                EVENTO  : INT;  (* Parâmetro 1 da CHF*)
                HORA    : INT;  (* Parâmetro 2 da CHF*)
                MINUTO  : INT;  (* Parâmetro 3 da CHF*)
                SEGUNDO : INT;  (* Parâmetro 4 da CHF*)
        END_VAR

        (* Insere o evento *)
        INSERIR_VALOR( MINUTO*256 + SEGUNDO );
        INSERIR_VALOR( HORA );
        INSERIR_VALOR( EVENTO );

END_PROGRAM
```

Conversão de valores

O módulo P-CONV.040 converte os valores de uma tabela de graus Fahrenheit para graus Celsius, armazenando os valores em outra tabela.

```
(*
        Converte as temeperaturas de °F para °C
*)

(* Funções ------------------------------------------------------------------------------- *)
(*
        Função: Converter

        Executa a conversão de uma unidade para outra. Retorna o valor convertido.
        Realiza a operanção utilizando o tipo REAL para obter percisão.
 *)

FUNCTION CONVERTER : INT
VAR_INPUT
        ENTRADA                 : REAL;
        MAXIMO_ENTRADA          : REAL;
        MINIMO_ENTRADA          : REAL;
        MAXIMO_SAIDA            : REAL;
        MINIMO_SAIDA            : REAL;
END_VAR

        (* Normaliza o valor de entrada*)
        CONVERTER :=    REAL_TO_INT( ENTRADA / (MAXIMO_ENTRADA - MINIMO_ENTRADA) *
                        (MAXIMO_SAIDA - MINIMO_SAIDA) );

END_FUNCTION


(* Programa ------------------------------------------------------------------------------ *)
PROGRAM P_CONV_040

        (* Variáveis *)
        VAR
                TEMPERATURAS_EM_CELSIUS       AT %TM0010[ 0 ]: ARRAY[ 1..100 ] OF INT;
                TEMPERATURAS_EM_FAHRENHEIT    AT %TM0011[ 0 ]: ARRAY[ 1..100 ] OF INT;
                I : INT;
        END_VAR

        (* Constantes *)
        VAR CONSTANT
                INICIO  : INT := 1;
                FIM     : INT := 100;
        END_VAR

        (* Converte as temperaturas de graus Celsius para graus Fahrenheit *)
        FOR I := INICIO TO FIM DO
                TEMPERATURAS_EM_FAHRENHEIT[ I ] :=
                        CONVERTER( TEMPERATURAS_EM_CELSIUS[ I ],
                        0, 100,
```

```
                              32, 232 );
          END_FOR;

END_PROGRAM
```

7. Apêndices

Palavras Reservadas

Na seqüência é apresentada a relação palavras reservadas pertencentes à Linguagem ST. Nem todas são utilizadas atualmente pela linguagem ST, mas já foram reservadas para futuras implementações.

```
AT
ARRAY OF
CASE OF ELSE END_CASE
INPUT1 INPUT2 INPUT3
OUTPUT1 OUTPUT2 OUTPUT3
INTERNAL_*
EXIT
FALSE TRUE
FOR TO BY DO END_FOR
FUNCTION END_FUNCTION
IF THEN ELSIF ELSE END_IF
REPEAT UNTIL END_REPEAT
RETURN
CONSTANT
VAR END_VAR
VAR_INPUT END_VAR
WHILE DO END_WHILE
BOOL
SINT USINT INT UINT
REAL
AND OR XOR NOT MOD
*_TO__** (conversão de tipos)
EN ENO
F_EDGE
FUNCTION_BLOCK
END_FUNCTION_BLOCK
PROGRAM WITH
PROGRAM END_PROGRAM
R_EDGE
READ_ONLY READ_WRITE
RESOURCE ON END_REPEAT
RETAIN
STRUCT END_STRUCT
TASK
TYPE END_TYPE
VAR_IN_OUT END_VAR
VAR_OUTPUT END_VAR
VAR_EXTERNAL END_VAR
VAR_ACESS END_VAR
WITH
BYTE WORD DWORD LWORD
DINT LINT UDINT U LINT
LREAL TRUNC
TIME DATE TIME_OF_DAY TOD
DATE_AND_TIME DT
ANY ANY_NUM ANY_REAL
 ANY_INT ANY_BIT STRING
 ANY_DATE
ABS SQRT LN LOG EXP SIN COS
TAN ASIN ACOS ATAN
SEL MAX MIM LIMIT MUX
LEFT RIGHT MID CONCAT INSERT
DELETE REPLACE LEN FIND
JMP CAL RET
ADD MUL DIV EXPT MOVE
SHL SHR ROR ROL
GT GE EQ LE LT NE
N R S L D P SD DS SL LD ST
ACTION END_ACTION
INITIAL_STEP END_STEP
STEP END_STEP
TRANSITION FROM TO
END_TRANSITION
```

8. Glossário

Glossário Geral

Algoritmo Seqüência finita de instruções bem definidas, objetivando a resolução de problemas.

Barramento Conjunto de sinais elétricos agrupados logicamente com a função de transferir informação e controle entre diferentes elementos de um subsistema.

Bit Unidade básica de informação, podendo estar no estado 0 ou 1.

Byte Unidade de informação composta por oito bits.

Ciclo de varredura Uma execução completa do programa aplicativo de um controlador programável.

Circuito de cão de guarda Circuito eletrônico destinado a verificar a integridade do funcionamento de um equipamento.

Código comercial Código do produto, formado pelas letras PO, seguidas por quatro números.

Controlador programável Também chamado de CP. Equipamento que realiza controle sob o comando de um programa aplicativo. É composto de uma UCP, uma fonte de alimentação e uma estrutura de E/S.

CP Veja controlador programável.

Default Valor predefinido para uma variável, utilizado em caso de não haver definição.

Diagnóstico Procedimento utilizado para detectar e isolar falhas. É também o conjunto de dados usados para tal determinação, que serve para a análise e correção de problemas.

Download Carga de programa ou configuração no CP.

E/S Veja entrada/saída.

Entrada/saída Também chamado de E/S. Dispositivos de E/S de dados de um sistema. No caso de CPs, correspondem tipicamente a módulos digitais ou analógicos de entrada ou saída que monitoram ou acionam o dispositivo controlado.

Estação de supervisão Equipamento ligado a uma rede de CPs ou instrumentação com a finalidade de monitorar ou controlar variáveis de um processo.

Hardware Equipamentos físicos usados em processamento de dados onde normalmente são executados programas (software).

IEC 61131 Norma genérica para operação e utilização de CPs. Antiga IEC 1131.

Interface Dispositivo que adapta elétrica e/ou logicamente a transferência de sinais entre dois equipamentos.

Interrupção Evento com atendimento prioritário que temporariamente suspende a execução de um programa e desvia para uma rotina de atendimento específica

kbytes Unidade representativa de quantidade de memória. Representa 1024 bytes.

LED Sigla para light emitting diode. É um tipo de diodo semicondutor que emite luz quando estimulado por eletricidade. Utilizado como indicador luminoso.

Linguagem Assembly Linguagem de programação do microprocessador, também conhecida como linguagem de máquina.

Linguagem de programação Um conjunto de regras e convenções utilizado para a elaboração de um programa.

Linguagem de relés e blocos Altus Conjunto de instruções e operandos que permitem a edição de um programa aplicativo para ser utilizado em um CP.

Lógica Matriz gráfica onde são inseridas as instruções de linguagem de um diagrama de relés que compõe um programa aplicativo. Um conjunto de lógicas ordenadas seqüencialmente constitui um módulo de programa.

MasterTool Identifica o programa Altus para microcomputador, executável em ambiente WINDOWS®, que permite o desenvolvimento de aplicativos para os CPs das séries Ponto, Piccolo, AL-2000, AL-3000 e Quark. Ao longo do manual, este programa é referido pela própria sigla ou como programador MasterTool.

Menu Conjunto de opções disponíveis e exibidas por um programa no vídeo e que podem ser selecionadas pelo usuário a fim de ativar ou executar uma determinada tarefa.

Módulo (referindo-se a hardware) Elemento básico de um sistema completo que possui funções bem definidas. Normalmente é ligado ao sistema por conectores, podendo ser facilmente substituído.

Módulo (referindo-se a software) Parte de um programa aplicativo capaz de realizar uma função específica. Pode ser executado independentemente ou em conjunto com outros módulos, trocando informações através da passagem de parâmetros.

Módulo C Veja módulo de configuração.

Módulo de configuração Também chamado de módulo C. É um módulo único em um programa de CP que contém diversos parâmetros necessários ao funcionamento do controlador, tais como a quantidade de operandos e a disposição dos módulos de E/S no barramento.

Módulo de E/S Módulo pertencente ao subsistema de entradas e saídas.

Módulo E Veja módulo execução.

Módulo execução Módulo que contém o programa aplicativo, podendo ser de três tipos: E000, E001 e E018. O módulo E000 é executado uma única vez, na energização do CP ou na passagem de programação para execução. O módulo E001 contém o trecho principal do programa que é executado ciclicamente, enquanto que o módulo E018 é acionado por interrupção de tempo.

Módulo F Veja módulo função.

Módulo função Módulo de um programa de CP que é chamado a partir do módulo principal (módulo E) ou a partir de outro módulo função ou procedimento, com passagem de parâmetros e retorno de valores. Atua como uma sub-rotina.

Módulo P Veja módulo procedimento.

Módulo procedimento Módulo de um programa de CP que é chamado a partir do módulo principal (módulo E) ou a partir de outro módulo procedimento ou função, sem a passagem de parâmetros.

Nibble Unidade de informação composta por quatro bits.

Octeto Conjunto de oito bits numerados de 0 a 7.

Operandos Elementos sobre os quais as instruções atuam. Podem representar constantes, variáveis ou um conjunto de variáveis.

PC Sigla para programmable controller. É a abreviatura de controlador programável em inglês.

Programa aplicativo É o programa carregado em um CP, que determina o funcionamento de uma máquina ou processo.

Programa executivo Sistema operacional de um controlador programável. Controla as funções básicas do controlador e a execução de programas aplicativos.

Software Programas de computador, procedimentos e regras relacionadas à operação de um sistema de processamento de dados.

Subsistema de E/S Conjunto de módulos de E/S digitais ou analógicos e interfaces de um controlador programável.

Tag Nome associado a um operando ou a uma lógica que permite uma identificação resumida de seu conteúdo.

UCP Sigla para unidade central de processamento. Controla o fluxo de informações, interpreta e executa as instruções do programa e monitora os dispositivos do sistema.

WD Sigla para cão de guarda em inglês (watchdog). Veja circuito de cão de guarda.